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

FIPA 2000: un modello per lo sviluppo di piattaforme interoperabili per software ad agenti

di Alberto Alvino

Tesi di diploma di laurea
svolta presso il Diploma in Informatica
dell' Università di Messina

Professore relatore
Prof. Antonio Puliafito

Messina
novembre 2000

Sommario

Parte 1 – Introduzione all’argomento

1.1 Concetti dell’evoluzione del software

1.1.1 L’evoluzione del software

La ricerca nel campo della computer science, oggi chiamata anche, ma in maniera più restrittiva, information tecnology, ha portato a diverse rivoluzioni nel settore. Ma più che di rivoluzioni bisogna parlare di idee rivoluzionarie, o innovative che dir si voglia, in quanto ognuna di queste ha impiegato il suo tempo per diffondersi. Verso la fine degli anni 50, aumentando consistentemente la complessità richiesta ai sistemi software, si diffusero i linguaggi di programmazione ad alto livello, la cui caratteristica era quella di “trasformare” il linguaggio con cui il calcolatore comprende le operazioni da compiere, in un linguaggio con proprietà più “umane”, un linguaggio cioè in cui a semplicissime e numerosissime istruzioni di spostamento di dati e operazioni aritmetiche si sostituiscono notazioni funzionali, costrutti logici e visione strutturata del programma. Iniziava così il processo di *astrazione* dei sistemi di programmazione. Verso la fine degli anni 60 un ricercatore della AT&T, Dennis Ritchie, progettò il linguaggio C. Questo si presenta come un linguaggio di alto livello con caratteristiche eccezionali: “è la migliore approssimazione possibile al linguaggio assembler” ([4]), da molti è infatti definito come linguaggio di medio livello; è strutturato ed incredibilmente coerente, come si vede dal limitato numero di parole chiave e dalla sintassi relativamente semplice; è tipizzato ma nonostante ciò estremamente flessibile; è basato sin

dall'inizio su una visione dinamica delle strutture dati; pur potendo essere potente con i suoi costrutti, permette di descrivere azioni ad un livello decisamente più basso degli altri linguaggi di alto livello. Soppianterà col tempo, almeno nella maggior parte degli utilizzi, linguaggi come Fortran, Algol o PL1. Grazie a queste caratteristiche sarà per la prima volta possibile realizzare il primo sistema operativo programmato interamente su linguaggio evoluto (cioè non assembler), Unix. In realtà Multics era già stato programmato con linguaggio evoluto, il PL1, ma non funzionò mai correttamente [12]. Fu infatti lo Unix di Ken Thompson, quasi interamente programato in C e con molte idee prese a prestito da Multics, a diffondersi e a diventare lo standard de facto, almeno nei sistemi hardware di medio-alto livello. Tra le idee base di Unix c'è la cooperazione tra programmi: per la prima volta lo sviluppatore software aveva un mezzo, datogli dal sottostante sistema operativo, per far comunicare direttamente due programmi in esecuzione contemporanea: era possibile costruire cioè più programmi semplici e gestibili facendoli cooperare in seguito ed in maniera diversa a seconda di ciò che si voleva ottenere, piuttosto che sviluppare un solo e complicato programma "monolitico". Nasceva così il concetto di *cooperazione*.

Continuando ad aumentare le richieste degli utenti e quindi il "peso" del software, la ricerca puntò sullo sviluppo di nuove metodologie di progettazione e programmazione: si poneva l'attenzione su come il software veniva realizzato, sul processo di produzione. Inoltre a questo punto erano passati una ventina d'anni dall'inizio dell'informatizzazione dei sistemi, e cominciavano a sentirsi tutti i limiti dei vecchi programmi: chiusi nella loro visione del mondo (cioè del loro ambito di funzionamento); difficili da modificare nel caso in cui le esigenze dell'utente cambiassero nel tempo; difficili da correggere nel caso in cui nel corso dell'utilizzo, magari a distanza di tempo, si scoprissero errori (o bug). La mancanza di tecniche efficaci di documentazione rendeva notevolmente arduo il compito di programmatori esterni al progetto originario di capire il codice sorgente cui si trovavano di fronte, ed eventualmente modificarlo. A vari anni di distanza dall'inizio dell'era informatica (parliamo qui degli USA) cominciava a denotarsi il bisogno della "manutenzione del software": il sotware non è statico (almeno quello utile), ma deve evolvere nel tempo in base alle esigenze di chi lo usa, deve essere aggiornato, corretto, e tutto questo deve essere fatto in maniera conveniente sotto il profilo di tutte le risorse umane, temporali, economiche. Questo era sostanzialmente il risultato di vent'anni di esperienza nella produzione del software. L'esperienza e la ricerca convergevano nell'idea di distinguere nel processo di produzione la fase di progettazione, quella fase in cui si definiscono le linee guida e la struttura che avrà il software, e la fase di programmazione, in cui, basandosi sul progetto, si sviluppano le varie parti di codice da mettere insieme secondo la struttura di cui sopra. Nascono allora concetti, oggi diffusisi anche in altri campi, come top-down o bottom-up.

Sempre in quel periodo, proprio per i motivi cui si è ora discusso, si comprese che la possibilità data dalla maggior parte dei linguaggi di programmazione (per non dire di tutti) di descrivere un sistema mediante funzioni non bastava a fornire la necessaria astrazione al progettista, che deve preoccuparsi del problema da un punto di vista macroscopico e non dei dettagli implementativi (cioè dei problemi microscopici). Il passo successivo fu quindi la nascità dei linguaggi modulari (p.es. ADA e Modula-2), che aggiungevano alle normali caratteristiche dei linguaggi strutturati dei costrutti per definire l'*astrazione* di modulo; possiamo sinteticamente definire quest'ultima come una collezione di funzioni tra loro correlate e/o interdipendenti. Il software viene quindi visto ora come una collezione di moduli (piuttosto che come una collezione di funzioni) cooperanti, ognuno dei quali contribuisce

nella sua parte. Anche qui quello che si ritrova è il concetto di *cooperazione*, ma da un punto di vista diverso: l'accento si pone sull'interno di un programma (chi guarda è lo sviluppatore), che è visto come l'insieme di più entità modulari ed in un certo senso "componibili", piuttosto che sull'ambiente in cui i programmi funzionano (qui è l'utente che guarda), che può o meno permettere la cooperazione tra tali programmi.
Ad un'ulteriore *astrazione* giungerà la ricerca nel corso degli anni 80, con la cosiddetta metodologia orientata agli oggetti (OO, Object-Oriented), che probabilmente costitisce il maggior passo in avanti compiuto dalla ricerca stessa in questo settore. Partendo dal linguaggio Simula67 (che per l'appunto risale al 1967), nato allo scopo di facilitare le simulazioni al calcolatore, si è notato che era conveniente per lo sviluppatore considerare non tanto le azioni da fare, ma piuttosto le entità, gli oggetti (da qui il nome) con cui avere a che fare. É così che si inverte il normale punto di vista di un programma: non un insieme di funzioni, cioè azioni (verbo) da compiere sulle strutture dati (sostantivo), ma un insieme di oggetti che hanno capacità di interagire, e cioè *cooperare* -si vedrà più avanti la differenza tra i due termini-, agendo su altri oggetti. Nascono SmallTalk, Eiffel, Objective-C, C++ e, più tardi, Java. Gli oggetti fondamentalmente sono entità che hanno uno stato interno (quindi posseggono delle caratteristiche rappresentate dalle strutture dati) e su cui è possibile svolgere –con terminologia tecnica, invocare– delle azioni –metodi–.
Si è voluto brevemente parlare di alcune delle idee che hanno fatto la storia dell'informatica, per trovare i concetti guida dell'evoluzione e della ricerca.

1.1.2 Astrazione

A ben guardare, i nuovi metodi di sviluppo del software non hanno aggiunto capacità in più, nè tantomeno nuove possibilità, almeno a livello teorico: non c'è niente che possa essere fatto da un programma OO e non da un programma scritto interamente in assembler o magari in codice macchina. Questo è fatto noto da quando Alan Turing parlò di macchina universale, nei tardi anni 40. Il problema dell'evoluzione del software è invece legato alla comprensibilità. Lo psicologo George Miller in un suo lavoro parla del fenomeno del "chunking" (letteralmente spezzettare) nella mente umana: si riferisce al fatto che questa è in grado di tenere presenti un numero limitato di concetti (pensieri, eventi, azioni, …) contemporaneamente per la costruzione di pensieri logici. Avendo, per esempio, un certo numero di oggetti (reali) uguali e visibili insieme con un solo colpo d'occhio, è possibile apprenderne il numero senza contarli solo se questi non sono più di sei o sette. Altrimenti bisogna "spezzettare" l'immagine e contare i pezzi. Per questa ragione conviene contare a tre a tre o a quattro a quattro, perché si fa maggior uso delle caratteristiche di comprensione della mente. Ed è per questo che avere milioni di righe di codice assembler porta ad una totale incomprensibilità di un sistema software, mentre al contrario focalizzare i pensieri su un insieme ridotto di oggetti (informatici) può portare alla totale comprensione dello stesso "in un solo colpo d'occhio". L'astrazione porta ad avere concetti ben più potenti ed espressivi rispetto a una banalissima operazione aritmetica. E dal momento che il software è il modo per rappresentare in una macchina il modello di un certo sistema umano, ecco che astrarre vuol dire portare il calcolatore più vicino alle problematiche umane.

1.1.3 Cooperazione

L'evoluzione dei sistemi di trattamento dell'informazione e la diversificazione delle piattaforme e dei servizi disponibili hanno reso sempre più indispensabile la cooperazione dei sistemi. A più livelli.

Appartiene decisamente al passato la classica applicazione software, intesa come un programma per elaboratore che svolge alcune operazioni a partire da dati in ingresso e dati memorizzati in precedenza, e restituisce dati in uscita oltre alla eventuale modifica dei dati memorizzati, con l'input-otput gestito da una più o meno funzionale interfaccia uomo-macchina. Un tale sistema chiuso si è rivelato insufficiente per la maggior parte dei bisogni: la visione, oggi, di un sistema complesso è necessariamente legata a diversi software in grado di interagire per ottenere le trasformazioni volute sui dati, le quali di volta in volta possono essere diverse, o semplicemente evolvere nel tempo. I singoli programmi devono quindi interfacciarsi ed interagire, cioè cooperare.

La tipologia di interazione con l'utente determina il modello con cui l'utente stesso può scambiare dati con il sistema informatico: al meccanismo di scambio dell'informazione attraverso posta elettronica (protocollo SMTP) l'utente si rapporterà sicuramente in maniera differente rispetto a quello basato su ipertesto (protocollo HTTP), e lo stesso vale per l'invio o la ricezione di blocchi di dati con tecnologie di tipo a flusso continuo (streaming) o trasferimento intero dei dati (p.es. FTP). E' quindi importante che un sistema completo sia in grado di provvedere ai bisogni dell'utente secondo il modello da questi preferito, e non è affatto detto che sia uno solo. Un sistema completo deve pertanto provvedere alla cooperazione di varie metodiche di comunicazione con l'utente.

E' andata aumentando, col tempo, la diversificazione delle piattaforme hardware e software sia in termini di tecnologie impiegate sia in termini di scala:

- elaboratori: si va dai grossi server aziendali con sistemi di data storage ai computer portatili o palmari (laptop), passando per supercalcolatori paralleli (Cray, Thinking Machines), workstation RISC e personal computer;
- reti: si va da linee di comunicazione WAN terrestri basate su fibra, a commutazione di pacchetto o ATM, a reti mobili (GSM e futuro UMTS), passando per reti locali, connessioni dial-up o DSL e canali di comunicazione satellitari o a microonde;
- sistemi operativi: si va da sistemi operativi multiutente (Unix e suoi cloni a norme POSIX, Windows NT) a sistemi per mainframe (VM, AS400), passando per sistemi monoutente (Windows, BeOS, MacOs), sistemi microkernel (Mach) e real-time (QNX), distribuiti e non, scalabili e non;
- applicazioni distribuite e comunicazione tra processi: si va da tecnologie di integrazione dei componenti software (CORBA, DCOM, Java RMI) a sistemi di messaggistica inter-applicazione (MAPI, Java Messaging System, MQ Series Enterprise Messaging), passando per chiamate di procedura remota (RPC), Unix IPC, Socket TCP, code messaggi, semafori e monitor, o i più semplici telnet e display remoto X-Windows;
- basi di dati: da DBMS aziendali ad oggetti (CA Jasmine) o relazionali (Oracle, Informix) a sistemi di piccole dimensioni (Microsoft Jet), passando per database distribuiti, data warehouse a multidatabase.

Una tale varietà di possibilità può esistere solo se è garantita l'interazione tra i singoli prodotti all'interno delle categorie: in tal caso infatti risulta possibile impiegare i prodotti dove più

conveniente per le loro caratteristiche intrinseche, mantenendo sempre l'apertura necessaria per ottenere un sistema unico e1 compatto. E questa è cooperazione.
Si ricorda che proprio su questi concetti si è sviluppato il concetto di internetwork, riuscendo a connettere tra loro sistemi hardware di rete profondamente diversi per concezione, ed è nato l'IP, Internet Protocol, proprio con un processo di astrazione sui protocolli di rete ([2]).
In generale un sistema informatico ottenuto con sottosistemi cooperanti è più simile al sistema umano che modellizza, semplicemente perché la cooperazione è parte integrante del modo di operare umano.

Si è ritenuto di dover introdurre in maniera esauriente questi concetti di base per arrivere a quelli che sono i motivi, le metodiche e le tecniche che hanno portato ad ideare un concetto come quello di "agente", prima, e a consentire lo sviluppo delle specifiche FIPA per piattafore basate sugli agenti, dopo.

1.2 – Introduzione al modello ad agenti

1.2.1 Definizione di agente software

Nella ricerca in campo informatico, che non si occupa di scoprire o dimostrare, ma piuttosto di pensare a nuovi concetti e modelli che possano poi avere una certa valenza in campo applicativo, capita spesso che di un nuovo concetto non ci siano definizioni precise ed univoche. Questo perché diversi ricercatori sono portati a guardare le cose da diversi punti di vista, ed è solo col tempo che si giunge alla formalizzazione precisa dei concetti e ad una convergenza sulle definizioni da adottare, soprattutto grazie all'esperienza maturata nell'applicazione del nuovo concetto. Così avvenne, come si è già detto, per il modello object-oriented, che a ben guardare nasce col Simula prima del 1970. La visione degli oggetti Simula, comunque, non era certo quella odierna. Fu negli anni 80 che fu precisato cosa si intendeva per object-oriented e cosa no, e fu data una definizione scrupolosa.
La situazione è ora la stessa nel caso degli agenti software, perché è il concetto di agente ad essere relativamente nuovo. Pur non essendoci una definizione formale universalmente accettata, è possibile capire cosa sia un agente a partire dai diversi punti di vista dei ricercatori che ne seguono lo sviluppo.
Il modello ad agenti costituisce un ulteriore grado di evoluzione del software, andando oltre il modello ad oggetti: supera quest'ultimo sotto il profilo dell'astrazione e della cooperazione. Si può dire ([10]) che un agente è un'entità software funzionante autonomamente e con continuità in un certo ambiente, spesso abitato da altri agenti ed altri processi. La definizione è corretta ma non esaustiva: posto in questi termini non è chiaro quale sia la differenza tra un agente ed un normale processo in esecuzione. La parola "agente" di per sé indica qualcuno (o qualcosa nel caso degli agenti software) che agisce per conto di qualcun altro nello svolgere un compito che gli è stato

assegnato. Nel nostro caso il termine è adatto perché un agente software svolge compiti per l'utente. A questa definizione alcuni ricercatori aggiungono che gli agenti, come l'uomo, hanno la facoltà di cooperare fra loro per costituire una "società di agenti", combinando le varie abilità di risolvere problemi ([6]). Ciò che distingue il modello ad agenti, infatti, è la maggiore capacità di cooperare rispetto ai modelli precedenti. La ricerca nel campo dei linguaggi formali e dell'intelligenza artificiale ha permesso la creazione di modelli di linguaggio capaci di avvicinarsi al linguaggio umano: non si tratta di emulare la complessità e la varietà che può avere una frase in linguaggio naturale, quanto di precisare termini con il loro significato semantico e attraverso una struttura logica poterli usare per descrivere conoscenza: si dice infatti che gli agenti, adoperando questo tipo di linguaggi, hanno la capacità di comunicare tra loro a "livello di conoscenza". Inoltre un agente è solitamente concepito per "esistere", e quindi possedere una identità ed uno stato, per un lungo periodo di tempo ([10]). Quest'ultima espressione è volutamente vaga: per "lungo" s'intende un periodo tale da far sì che l'agente stesso abbia modo di accumulare esperienza, imparare e migliorarsi, crearsi una sua idea del mondo, ed avere, in un certo qual modo, una sua personalità, e quindi un suo modo di rapportarsi con gli altri agenti, di reagire agli eventi. Questo può avvenire proprio in virtù della potenza del nuovo meccanismo di comunicazione. E' chiaro che siamo di fronte ad una potente astrazione, che avvicina notevolmente al pensiero umano la modellizzazione di un sistema software. Inoltre gli agenti sono più dinamici, nel senso tecnico del termine: proprio perché un agente ha certe facoltà di imparare e quindi di inferenza, esso ha meno bisogno di conoscenza a priori e i suoi compiti possono essere specificati in maniera astratta.

Molti ricercatori includono nelle facoltà degli agenti anche la mobilità: un agente mobile è in grado di migrare da un posto ad un altro di sua scelta, per sua propria decisione, e per tale ragione non può che essere più intelligente in quanto più simile all'uomo. A tale proposito in 2.8.4 verrà fatta una distinzione chiarificatrice tra le diverse accezioni del termine "agente mobile".

Possiamo stilare un'elenco delle principali caratteristiche di un agente, fermo restando che ne esistono di secondarie e che non sono tutte indispensabili ("The term agent might best be viewed as an umbrella term that covers a range of more specific and limited agent types" [5]):

- reattività
- autonomia
- attitudine alla collaborazione
- facoltà di comunicazione a "livello di conoscenza"
- capacità di inferenza
- continuità temporale
- personalità
- adattamento
- mobilità

In base alle caratteristiche di cui un agente dispone è possibile classificarlo in una o più di queste categorie([5]):

- *agenti cooperanti*: qui l'enfasi è posta sull'atonomia e la cooperazione di più agenti
- *agenti d'interfaccia*: sono gli agenti che dialogano con l'utente: l'enfasi è posta sull'autonomia e sulla capacità di apprendimento;

- *agenti mobili*: agenti capaci di percorrere reti per trovare informazioni richieste dall'utente, ed in seguito tornare al nodo di partenza;
- *agenti d'informazione*: manipolano e conservano informazioni da sorgenti distribuite;
- *agenti reattivi*: rispondono con una certa politica a certi eventi che avvengono nel loro ambiente, senza però avere un proprio modello simbolico dell'ambiente stesso;
- *smart agent*: i programmi di intelligenza artificiale spesso hanno proprietà di apprendimento e, per l'appunto, di intelligenza: programmi di questo genere, dotati di capacità di comunicazione come quelle sopra descritte, possono essere considerati agenti a tutti gli effetti.

1.2.2 MAS

La caratteristica principale di un agente è la capacità di interagire con altri agenti: la conseguenza è che il suo ambiente ideale è quello in cui esso può coesistere con una popolazione di altri agenti che renda possibile la reale interazione. Un sistema che realizzi questo ambiente è detto Multi Agent System (MAS), ed è a questo termine che ci si riferisce quando si parla di "piattaforma ad agenti", o "agent platform", da cui l'acronimo AP. Proprio dal coesistere in uno stesso ambiente di più agenti se ne deriva che ogni agente deve possedere solo un sottoinsieme delle caratteristiche precedentemente viste, associate al concetto di agente generico. Di conseguenza in un MAS dovranno essere presenti due o più agenti di diverso tipo, cioè appartenenti a diverse categorie.
Il problema principale nello sviluppare un MAS sta nel fatto che il termine "agente" è genericamente definito come un "pezzo di software". In un MAS dunque ([6])

- un agente può essere codice eseguibile come processo nativo su un certo microprocessore ed un certo sistema operativo, per esempio un programma compilato a partire da un sorgente ad alto livello con librerie collegate staticamente o dinamicamente, o un oggetto COM o un programma LISP autocontenuto, oppure può essere eseguito da un interprete, come nel caso di un componente Java, che richiede o hardware apposito all'esecuzione dei byte-codes oppure una Java Virtual Machine, di un oggetto SmallTalk o di un programma script Perl, Python, TCL o Scheme, che richieda l'apposito sottosistema, e quant'altro;
- gli agenti possono interfacciarsi e comunicare attraverso uno o più canali di comunicazione, o più precisamente di *trasporto*, hardware/software tra quelli oggi disponibili (di cui si è velocemente parlato in 1.1.3), con uno dei linguaggi adatti alla rappresentazione della conoscenza, e le loro interazioni possono basarsi su modelli diversi (scambio informazioni, richieste di servizi, delega di alcuni compiti, …);
- un agente deve essere in grado di localizzare un altro agente ben preciso con il quale voglia sostenere un dialogo;
- (dalla precedente) deve essere previsto un sistema di identificazione univoco (cioè identificatori globali);
- deve esserci un ambiente sicuro, tale che più agenti possano scambiarsi informazioni confidenziali;
- se si vuole un sistema veramente aperto, in grado quindi di comunicare all'esterno, deve essere previsto un modo per accedere a software non ad agenti;

- può essere prevista la capacità di mobilità da un luogo ad un altro della piattaforma o tra più piattaforme dello stesso tipo.

Il MAS è solitamente visto come uno strato di software sopra il quale funzionano (e del quale utilizzano i servizi) gli agenti. Ma questo strato può essere posizionato a livelli diversi nella stratificazione software di un sistema: a livello di sistema operativo, se gli agenti sono realizzati come codice eseguibile; a livello di Virtual Machine Java (quindi sopra il s.o. o addirittura su firmware, quando si è in presenza di hardware adatto, ossia una macchina fisica Java); a livello applicazione, se ad eseguire l'agente è un interprete; oppure a livello di singolo thread all'interno del processo costituito dall'AP.

Progettare una piattaforma ad agenti implica quindi numerose scelte che poi andranno a costituire la base dello sviluppo (dunque anche i pregi ed i difetti) degli agenti stessi su quella piattaforma, e ad influenzare i rapporti tra la piattaforma in questione e le altre. Va detto inoltre che quando ci si riferisce ad una piattaforma o a un luogo della piattaforma, non si intende specificare una rete o una singola macchina, in quanto la grandezza in termini fisici della piattaforma ed il modo in cui è suddivisa viene specificata solo dal progettista dell'AP.

Nessun vincolo è dato alla realizzazione interna dell'agente, che potrà essere (sempre limitatamente alla tecnologia di realizzazione scelta dal progettista del MAS) basata su linguaggi a basso o alto livello, strutturata o meno, oppure object-oriented. Data la superiorità di quest'ultima, molti agenti sono, e saranno, sviluppati con questa metodologia. Può essere utile chiarire perché un agente è più di un oggetto. Un oggetto si limita ad invocare i metodi di altri oggetti, oppure ad essere invocato per compiere un'azione sulla quale non ha possibilità di decisione o discernimento. Al contrario, un agente è un'entità autonoma in grado di "ragionare" su quello che gli viene chiesto. Semplificando, un agente "può dire no" [1], ed è per questa ragione che il modello ad agenti supera anche il modello client-server. Un oggetto esiste come ente attivo solo durante l'esecuzione di un suo metodo, altrimenti è immobile ed immutato, diversamente dagli oggetti reali che esistono ed interagiscono con la realtà indipendentemente dal fatto che lo si voglia o no. Al contrario, gli agenti sono in esecuzione costante, ed hanno perciò una continua esistenza attiva con facoltà di reattività e di socialità, cioè hanno quella capacità di autonomia non rappresentabile neanche nei modelli ad oggetti con supporto di multithreading.

1.2.3 L'interoperatività degli agenti

Precedentemente si è visto come due siano i concetti principali dell'evoluzione del software: astrazione e cooperazione. Si vuole qui andare più in dettaglio sulle relazioni tra i due concetti, chiarire alcune definizioni ed applicare il tutto al modello ad agenti [11].

Come visto in 1.1.3, la cooperazione è possibile a vari livelli. Questi non sono altro che differenti gradi di astrazione del processo di interazione tra più enti. Partendo dal caso di una conversazione verbale umana, vediamo che sono necessari un segnale ed un mezzo nel quale il segnale si muova affinchè possa esserci la conversazione. Il mezzo è chiaramente l'aria, il segnale è invece costituito dalle onde sonore. Siamo qui ad un livello di astrazione davvero basso, poiché a tale livello potremmo considerare comunicazione qualsiasi suono disarticolato, ma la conversazione che vogliamo sostenere è di ben altro tipo. Per conversare bisogna parlare la stessa lingua ed avere un vocabolario comune. Una parola è un insieme di suoni che implicitamente correliamo ad un

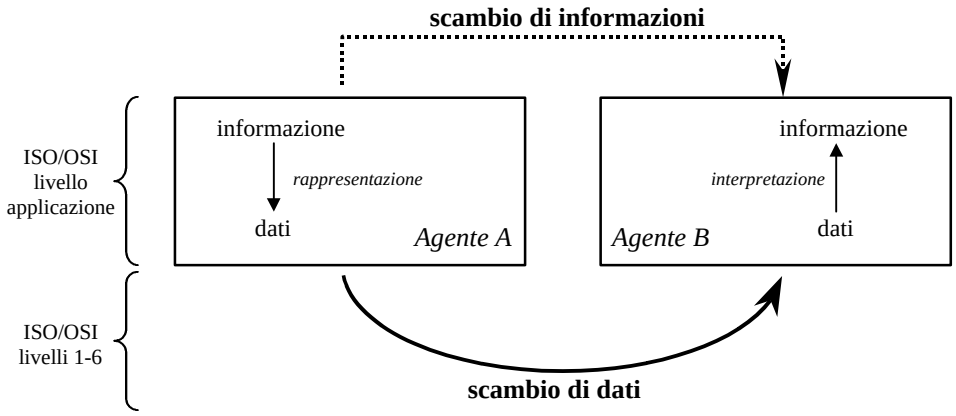


Fig. 1 [11]: il mezzo con cui si comunica (ISO/OSI 1-6) permette lo scabio di dati. Lo scambio di *informazione* avviene attraverso i passaggi di rappresentazione e di interpretazione. Anche il *dato* costituisce comunque una astrazione rispetto al semplice *segnale*.

concetto, è un simbolo di quel concetto, dotato quindi di una sua semantica, e considerare il concetto vuol dire astrarre rispetto al semplice insieme di suoni. Lo stesso vale per una frase, in cui si utilizzano strutture sintattiche di un certo linguaggio per correlare parole in modo tale da avere una certa semantica. Infine, gli stessi suoni sono insiemi di segnali adatti (in relazione al mezzo, quindi in questo caso all'aria) a scandire le parole. La conversazione è quindi una profonda astrazione, essendo frutto di milioni di anni di evoluzione dell'intelligenza umana. Una differenza fondamentale, come si è visto, sta nella differenza tra simbolo e concetto: è la stessa che passa tra dato e informazione. Volendo comunicare, un agente vorrà esprimere informazione ad un altro agente, e per poterlo fare deve concretizzare in simboli i concetti da esprimere: la trasformazione da informazione a dato viene chiamata *rappresentazione*. Tali dati saranno poi inviati sul mezzo di comunicazione prescelto ed in seguito raccolti dal destinatario (non obbligatoriamente uno solo). Quest'ultimo avrà il compito di ricavare dai simboli i concetti espressi: il ricavare l'informazione dai

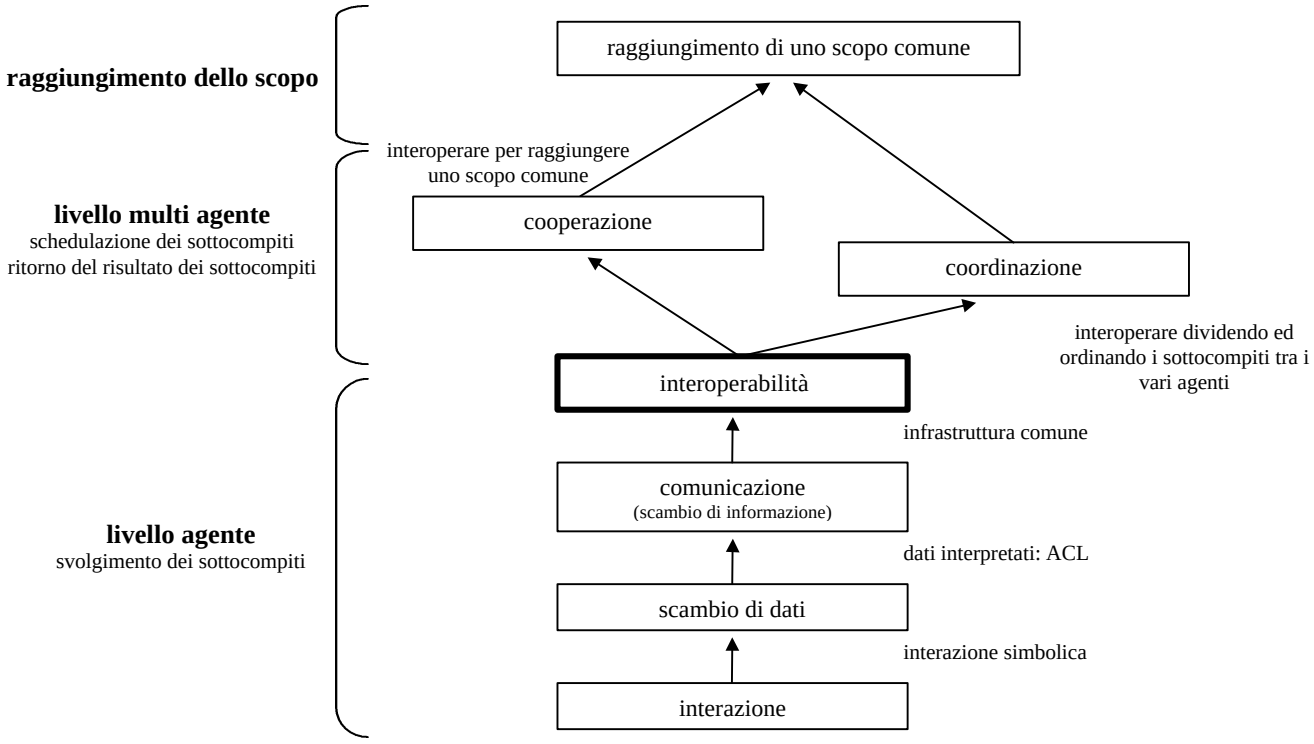


Fig. 2 [11]: scala dei livelli del processo di astrazione della comunicazione: dalla semplice interazione (scambio di segnali), allo svolgimento di compiti in modo coordinato e cooperante al fine di ottenere il raggiungimento di uno scopo comune

dati è detto *interpretazione*.
Lo standard ISO Open System Interconnection (ISO/OSI) stabilisce in maniera esaustiva e formale quali siano i passaggi (quindi i livelli) necessari affinchè si possa stabilire comunicazione tra due o più sistemi. Gli agenti non sono altro che i sistemi appena considerati, per cui giacciono al di sopra dei livelli del modello OSI e lo sfruttano per compiere comunicazioni. Possiamo finalmente definire il termine *comunicazione* come "lo scambio d'informazione", mentre per *interazione* ci si riferisce al fatto che un ente riceva un qualsiasi segnale da un altro: uno scambio di dati quindi non è altro che una interazione simbolica.
Concetto chiave a questo punto è l'*interoperatività* che si può instaurare tra agenti in grado di comunicare, e che può essere definita come la capacità di lavorare insieme senza bisogno di soluzioni ad hoc: è ovviamente un'ulteriore astrazione relativa al fatto che la comunicazione sia espressamente ideata per scambiarsi informazioni sulle funzionalità e le potenzialità degli agenti. Ed è attraverso l'interoperatività che sono realizzabili la *cooperazione*, cioè l'interoperabilità allo scopo di raggiungere un obiettivo comune, e la *coordinazione*, cioè la capacità di interoperare nel suddividere un compito complesso in più compiti semplici realizzabili dagli agenti.
Affinchè sia possibile l'interoperatività, è necessario che tra gli agenti siano comuni tutti i livelli sottostanti all'interoperatività stessa. Devono perciò esistere: a) una infrastruttra comune (la piattaforma); b) un linguaggio per le comunicazioni adatto, uno dei cosiddetti Agent Communication Language (ACL).

1.2.4 ACL

Come abbiamo già visto ([1.2.1]), è principalmente grazie alle nuove capacità dei linguaggi ACL che il nuovo modello di riferimento nello sviluppo dei sistemi software è quello ad agenti. Osserviamo alcune delle caratteristiche più generali di tali linguaggi.
La prima differenza che è possibile riscontrare in uno di questi linguaggi consiste nel fatto che tutte le comunicazioni avvengono in maniera *peer-to-peer* (pari a pari), il che vuol dire che tutti gli agenti sono messi su uno stesso "piano sociale", a differenza dei modi di interazione tipo client-server. Questo fa sì che la comunicazione possa essere considerata come uno scambio di messaggi, perché nel termine messaggio non è intrinseca alcuna superiorità o inferiorità di un interlocutore rispetto all'altro: *messaggio* è infatti il termine con il quale ci si riferisce alla singola comunicazione di un agente con un altro.
Le comunicazioni tra agenti si basano sulla *speech-act theory* [9]: questa teoria è stata sviluppata da filosofi e linguisti allo scopo di dividere in categorie (ed in maniera esaustiva) le frasi di un discorso parlato umano: per esempio, una frase può essere detta per informare, richiedere od offrire, proporre o confermare, esser d'accordo… La categorizzazione dei discorsi umani in maniera precisa è stata considerata la base sulla quale sviluppare un linguaggio formale più simile a quello umano. Un messaggio dunque apparterrà ad una singola categoria (detta Communicative Act o CA).
Gli ACL devono essere in grado di esprimere la conoscenza: devono cioè essere in grado di riferirsi a termini che abbiano una semantica precisata e metterli in relazione attraverso strutture logiche. Linguaggi pensati per questo scopo, ed a prescindere dal discorso sugli agenti, sono per esempio il KQML (knowledge querying meta language) e l'ARCOL. Solitamente gli ACL si basano su linguaggi come questi, già sperimentati in altri campi, invece di creare nuove sintassi apposite. In

particolare ([6]) il FIPA ACL (di cui si parlerà diffusamente nel seguito) si basa principalmente sull'ARCOL, poiché si è ritenuto che il KQML non avesse una semantica formale universalmente accettata e che desse un approccio troppo restrittivo alla comunicazione tra agenti, legando strettamente tra loro alcuni CA dal punto di vista della successione dei messaggi. Ma il linguaggio scelto influenza il modo in cui i termini potranno essere combinati tra loro per esprimere una certa conoscenza, non la scelta ed il significato stesso dei termini. È necessario definire delle ontologie. Il significato del termine "ontologia" è di matrice prettamente filosofica, ma rientra nel discorso poiché in una delle sue accezioni vuole indicare "la cosa in sé", ed è scelto per indicare la relazione tra il simbolo ed il concetto. Semplificando: affinchè possa esserci comunicazione, non basta avere un vocabolario comune, ma bisogna intendersi sulla semantica di ogni simbolo termine, e l'ontologia è quella entità che lega le due cose. Come se non bastasse, la conoscenza, e quindi i termini che la esprimono, è legata al *dominio* che si sta considerando. Possiamo definire informalmente "dominio" come l'argomento del discorso. Bisogna infatti definire una serie di ontologie, ognuna relativa ad un dominio, cioè ad un campo della conoscenza, e questo implica il problema di come dividere, o catalogare, la conoscenza. Ma proseguire qui il discorso sarebbe lungo e fuori luogo.

1.3 – FIPA

Si è già parlato della varietà di ambienti, di modi e di tecniche su e con cui sviluppare una piattaforma ad agenti, cioè un MAS. Sostanzialmente il modello ad agenti nasce in un periodo di forti diversificazioni in tutti i settori legati all'elaborazione automatica, periodo nel quale la diffusione dell'elaborazione automatica stessa si è estesa pressocchè ad ogni campo dell'attività umana e nel quale quindi la cooperazione tra i vari sistemi hardware/software è d'obbligo se si vuole che l'informazione possa viaggiare verso l'utente che la desidera e nel modo che desidera senza limiti di spazialità o temporalità (e questo è, almeno teoricamente, il fine ultimo dell'informatica). Inoltre, il modello ad agenti non chiarisce quali debbano essere i dettagli tecnici relativi al modello stesso (di cui al punto 1.2.2): come può sapere un agente con chi comunicare per avere svolto un certo servizio? Ed una volta saputo, come trovarlo? Qual è esattamente il linguaggio da lui compreso? Queste sono altre variabili che il progettista del MAS deve tenere in considerazione. E le variabili sono quelle che incidono sul numero di possibili combinazioni: in [11] è possibile trovare notizie su 11 MAS basati su Java senza supporto per codice mobile, 6 con supporto per codice mobile, e ben 19 MAS non basati su Java. E l'elenco non è esaustivo, oltre al fatto che queste piattaforme coprono solo un range parziale di tutte le possibili configurazioni.
Nasce quindi FIPA: Foundation for Intelligent Phisical Agents. È, come recita la prefazione di ogni documento da questa rilasciato, una organizzazione internazionale dedicata a promuovere la diffusione del modello ad agenti attraverso lo sviluppo di specifiche di dominio pubblico che consentano l'interoperatività tra agenti e applicazioni basate sugli agenti. FIPA si occupa quindi, dal 1997, di produrre delle specifiche che è possibile tenere a mente durante la fase di progetto di un MAS, al fine di garantire un certo grado di interoperabilità tra piattaforme diverse. Sostanzialmente è compito delle specifiche FIPA isolare il livello MAS ed i livelli superiori, dai livelli inferiori:

hardware di elaborazione, hardware di rete, canali e protocolli di comunicazione. Una AP viene dunque detta “conforme alle specifiche FIPA” se si attiene al modello e alle scelte indicati da tali specifiche. Quello di cui non si occupano le specifiche FIPA è il come realizzare internamente i singoli agenti: questo è infatti compito dello sviluppatore applicativo che realizzerà sistemi software basati sul modello ad agenti e funzionanti sulla piattaforma conforme alle specifiche FIPA.

FIPA definisce un suo ACL per lo scambio di messaggi tra agenti. Tale linguaggio può essere scomposto in cinque parti [6]:

- i communicative act: come si è detto precedentemente i CA permettono di identificare in quale categoria del discorso si inserisce un messaggio (richiesta, informazione, proposta, conferma);
- le informazioni di supporto: il messaggio conterrà informazioni quali il mittente ed il destinatario, il mezzo di comunicazione usato per il trasporto, il contesto del discorso nel quale vine inviato tale messaggio;
- protocolli di interazione (il termine non si riferisce ai protocolli di rete): permettono di definire certi “pattern” standard di messaggi durante una comunicazione, o, se si vuole, definiscono una sequenza di messaggi con precisi CA, che rappresentano un modello di una completa conversazione. Ad esempio, il protocollo “fipa-request” prevede che un agente richieda una azione ad un altro con CA “request” e che quest’ultimo risponda con un messaggio con CA “refuse” (rifiutato) se l’azione è impossibile o “agree” (cioè d’accordo) se l’azione è possibile, e, in quest’ultimo caso, terminata l’azione invii un ulteriore messaggio con CA “inform” per comunicare il successo o con CA “failure” (fallimento) per comunicare il fallimento dell’azione;
- i content language: FIPA affida la vera e propria comunicazione ad un sottostante linguaggio: è con la grammatica del content language, e con la sua semantica, che viene espresso il contenuto (per l’appunto) del messaggio; FIPA difinisce con precisione quali sono i content language che è possibile usare in una piattaforma conforme alle sue specifiche;
- le ontologie: un messaggio deve definire l’ontologia cui appartengono i termini che si ritrovano all’interno, cioè, semplificando, deve definire il vocabolario ed il significato semantico dei termini e concetti usati nelle espressioni del contenuto.

Le caratteristiche delle specifiche FIPA verranno esaminate nella parte dedicata.

I membri del FIPA fanno parte di oltre quaranta aziende, come British Telecom, IBM, Nortel Networks, Hewlett-Packard, Siemens, Alcatel, Toshiba e CSELT, diffuse in ogni parte del mondo. Di organizzazioni simili ne esistono già diverse, prime tra tutte il W3C (WorldWideWeb Consortium) e l’OMG (Object Management Group), rispettivamente rivolte allo sviluppo del web (e tecnologie associate) e alla diffusione del modello ad oggetti. Il loro compito è quello di proporre standard aperti con un largo consenso di industrie, e la loro produttività è da considerarsi un successo nell’ambito della cooperazione internazionale, contrapposta alla prepotenza di aziende che continuano tutt’oggi a sostenere, con la forza del mercato, tecnologie chiuse e proprietarie.

Parte 2 – Le specifiche FIPA 2000

2.1 – Una visione d'insieme: il sistema in generale

FIPA è un organismo, operante dal 1996, che ha prodotto tre serie di documenti di specifica, ognuna delle quali presenta un insieme di migliorie e/o di aggiunte rispetto alla serie precedente. In correlazione con l'anno di rilascio, tali serie prendono il nome di FIPA97, FIPA98 e FIPA 2000.
I vari argomenti su cui vertono i documenti di specifica hanno preso strade di evoluzione diverse. Con le specifiche FIPA 2000, inoltre, si hanno cambiamenti sia di ordine pratico sia di ordine concettuale. In effetti il passaggio da FIPA97 a FIPA98 è consistito più nell'aggiunta di ulteriori caratteristiche che non nella modifica di caratteristiche già proprie della versione precedente. È scopo di questo paragrafo cercare di dare un'idea generale sui principi fondamentali delle precedenti specifiche. Questo sia per dare un punto di partenza a chi non è finora entrato nel merito delle specifiche FIPA, sia per offrire un punto di attacco per chi già conosce le specifiche precedenti.
In molte pubblicazioni il principio di funzionamento dell'architettura FIPA, e quindi di tutte le piattaforme reali ad essa conformi, è esemplificato da un solo schema. Lo schema di fig. 3 mette in rilievo i componenti principali dell'architettura ed i rapporti, in via generale, tra questi.
Effettivamente lo schema della figura illustra per grandi linee il *modello di riferimento* FIPA: in particolare il modello di riferimento delle specifiche FIPA97 e FIPA98. Ciò che la figura non è in grado di rappresentare è la composizione, il livello (nella stratificazione software) di implementazione, i dettagli del singolo elemento, nonché i modi e le metodiche di interazione esistenti tra questi.

Ovviamente una piattaforma è creata per offrire supporto agli agenti che vi operano al di sopra. Nel modello FIPA il generico agente comunica con altri agenti mediante il FIPA ACL (l'ACL standard definito da FIPA). L'Internal Platform Message Transport è l'elemento della piattaforma che ha il compito di trasportare i messaggi dall'agente mittente all'agente destinatario, all'interno della piattaforma. Facendo una similitudine con i sistemi hardware, esso costituisce il bus interno della piattaforma. Ed è anche la base portante della piattaforma stessa. L'Agent Management System è l'elemento della piattaforma il cui scopo è quello di registrare tutti gli agenti operanti nella piattaforma insieme con il relativo indirizzo, utile per la loro localizzazione, oltre a quello di supervisionare la funzionalità e l'accesso alla piattaforma da parte degli agenti. Le specifiche FIPA97 e FIPA98 prevedono che questo componente sia implementato come agente, ed infatti le comunicazioni con esso avvengono tramite l'ACL. Si vedrà che le specifiche FIPA2000 sono a questo riguardo più flessibili. Quanto detto vale anche per il Directory Facilitator: anche per questo componente è prevista, nelle prime versioni di specifica, un'implementazione a livello di agente. Lo scopo del Directory Facilitator è quello di offrire un archivio pubblicamente accessibile delle funzionlità e dei servizi svolti dagli agenti operanti sulla piattaforma.

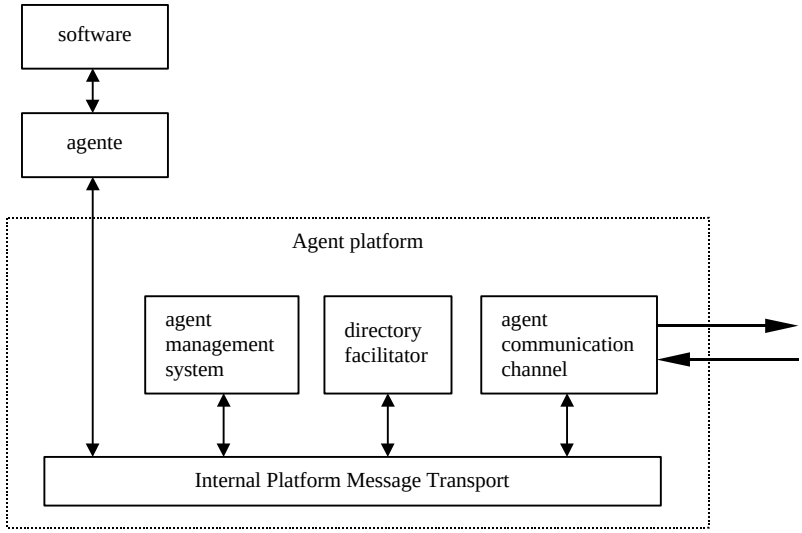


Fig. 3: la rappresentazione più comune del modello di riferimento FIPA

Il modello di riferimento prevede che un agente A in cerca di un altro agente con opportune caratteristiche si rivolga al Directory Facilitator presentandogli l'elenco di tali caratteristiche. Il Directory Facilitator, previa ricerca tra le informazioni di cui è in possesso, comunicherà ad A il nome degli (perché possono anche essere più d'uno) agenti con i requisiti richiesti. A dovrà in seguito, tramite una l'Agent Management System, collegare il nome con gli indirizzi effettivi cui poter raggiungere gli agenti di cui richiederà i servizi, o , in generale, con cui comincierà il dialogo.
Il modello richiede pertanto che ogni agente

- registri il suo nome ed indirizzo presso l'Agent Management System;
- registri i servizi che è capace di espletare, ed altre sue funzionalità in genere, presso il Directory Facilitator.

Il Directory Facilitator agisce quindi come un broker di servizi e rende possibile, per un agente, trovare nell'universo degli agenti un secondo agente con cui interagire per raggiungere un certo scopo.

L'Agent Communication Chanel è un l'elemento, nelle specifiche precedenti FIPA2000 implementato anch'esso come agente, che permette lo scambio di messaggi tra agenti operanti su piattaforme diverse. Il modo in cui questo è realizzato è piuttosto complesso e gli sono stati dedicati appositi paragrafi. Ciononostante si può dire che il dover mandare un messaggio ad un agente con l'intermediazione di un altro agente, appunto l'Agent Communication Chanel, è stato un fattore di critica nei confronti delle precedenti specifiche, per l'ovvia ragione di introdurre un netto collo di bottiglia nelle comunicazioni interpiattaforma.
Soprattutto per questo fattore, le specifiche 2000 hanno il vantaggio di non obbligare l'implementatore a realizzare l'Agent Communication Chanel a livello di agente, lasciandolo libero di scegliere altri mezzi di implementazione, quali per esempio API interne.
Nel modello FIPA un agente è per definizione collegato con l'Internal Platform Message Transport (che nelle specifiche 2000 prende nome di Message Transport System), che gli consente di scambiare messaggi in modo standard con altri agenti della stessa piattaforma o, con l'intermediazione dell'Agent Communication Chanel, con agenti di piattaforme diverse. Il modello prevede inoltre che un agente possa collegarsi con altri sistemi di comunicazione e di gestire i dettagli dell'uso di tale sistema. Questo approccio permette non solo il collegamento tra due agenti esterno ai meccanismi di piattaforma, ma soprattutto permette ad un agente di collegarsi a software non basato sul modello ad agenti, non operante cioè su alcuna AP e pertanto non avente alcun Internal Message Transport System a disposizione. Le comunicazioni tra un agente e tale software non saranno basate sull'ACL.

2.2 – Organizzazione delle specifiche FIPA[1]

Il primo rilascio di specifiche da parte di FIPA risale al 1997 con il nome di FIPA97. La composizione di tali specifiche è molto semplice. Questi sono i documenti tecnici previsti:

- Agent Management
- Agent Comunication Language
- Agent Software Integration

Il primo concerne la definizione di uno standard aperto per accedere ai servizi di gestione degli agenti, come per esempio l'identificazione e la localizzazione degli agenti. Il secondo si occupa di definire un preciso linguaggio di comunicazione tra gli agenti. Il terzo indica come sia possibile interfacciare una piattaforma, costruita conformemente ai precedenti documenti, al software non "ad agenti". Con l'uscita delle specifiche FIPA98 l'impostazione rimane uguale, ma vengono aggiunti i seguenti documenti:

- Human-Agent Interaction
- Agent Management Support for Mobility
- Ontology Service

[1] Tutte le specifiche ad oggi rilasciate da FIPA possono essere trovate presso il sito web dell'organizzazione all'indirizzo: http://www.fipa.org/repository/index.html

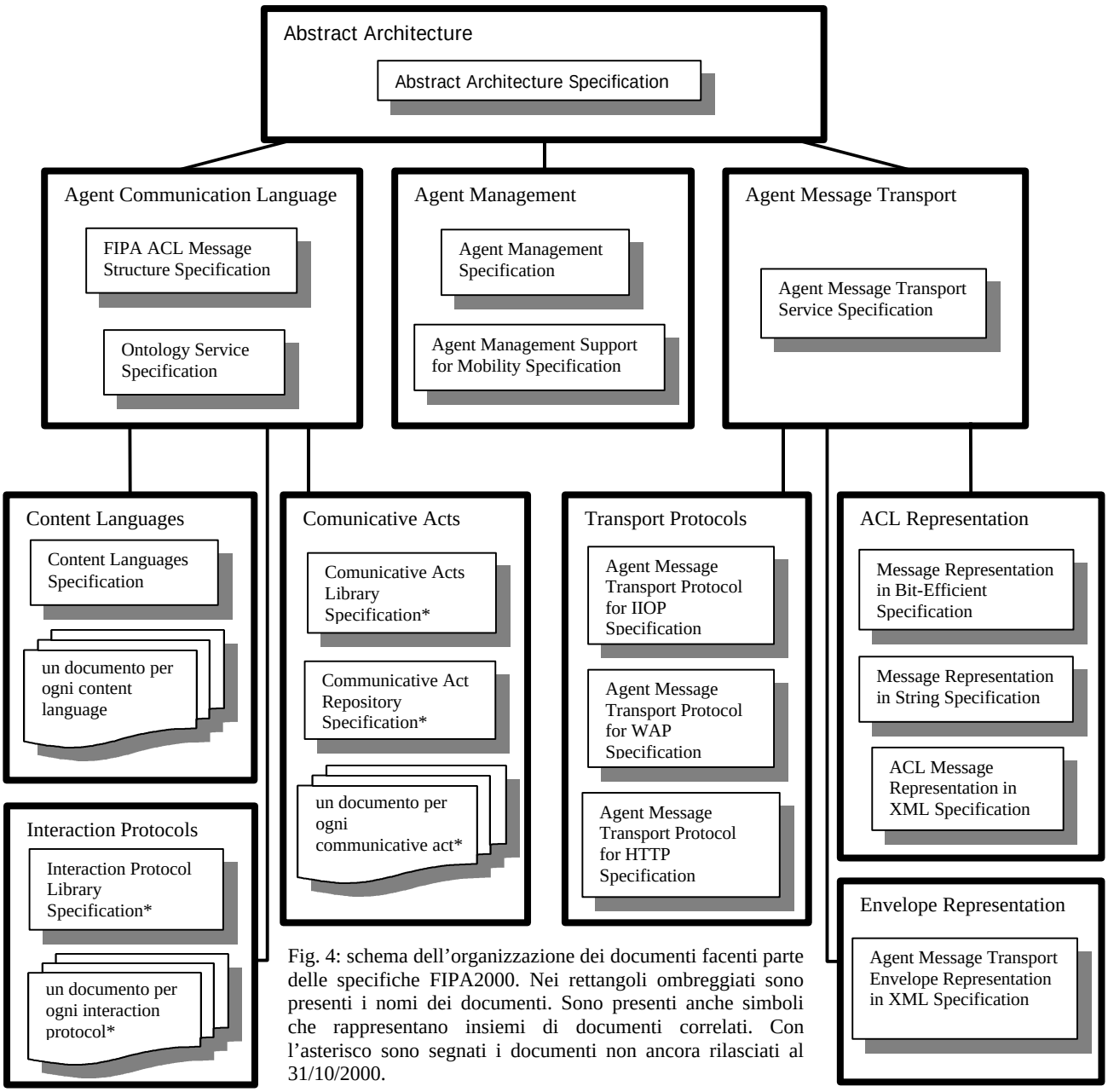


Fig. 4: schema dell'organizzazione dei documenti facenti parte delle specifiche FIPA2000. Nei rettangoli ombreggiati sono presenti i nomi dei documenti. Sono presenti anche simboli che rappresentano insiemi di documenti correlati. Con l'asterisco sono segnati i documenti non ancora rilasciati al 31/10/2000.

- Agent Security Management

Il primo descrive un modo standard per creare un'interfaccia uomo-macchina basata sugli agenti. Il secondo si occupa di rendere possibile in una piattaforma FIPA la mobilità degli agenti software. Il terzo concerne le tecnologie necessarie ad implementare un servizio di gestione delle ontologie; un servizio, quindi, che renda possibile l'accesso ad ontologie pubbliche, il loro mantenimento (aggiornamento), e quant'altro. Nel quarto si affrontano i punti chiave della sicurezza nei servizi di gestione della piattaforma e le modalità di una comunicazione sicura tra agenti.

Con l'uscita delle specifiche FIPA2000 l'organizzazione dei documenti subisce notevoli cambiamenti. In parte anche per il fatto che si adotta un metodo di produzione delle specifiche più simile ai dettami dati dall'IETF, l'Internet Engeneering Task Force. I documenti vengono scorporati in documenti più piccoli e più settoriali, viene introdotto un documento per la definizione di una architettura astratta, vengono dedicati documenti ad ognuno dei content language e per ognuno dei protocolli standard prevesti dall'architettura, vengono specificati i modi in cui una piattaforma FIPA

possa utilizzare sottostanti servizi di comunicazione, come l'Internet Inter Orb Protocol dell'OMG o il Wireless Application Protocol. Uno schema indicante le relazioni tra i documenti FIPA è quello in fig. 4. Nella figura non è presente il documento "Agent Software Integration Specification", che nella struttura discende direttamente dall'architettura astratta, e specifica i modi di interazione possibili tra un agente della piattaforma e software esterno non basato su agenti. Alcuni settori coperti dalle precedenti specifiche vengono abbandonati, poiché si ritiene che non esista ancora una visione sufficientemente completa e coerente degli argomenti trattati. I documenti di specifica dedicati a tali settori vengono considerati obsoleti e non sono rimpiazzati. Tali documenti sono: Human-Agent Interaction e Agent Security Management.

FIPA ha tracciato, fin dall'inizio, il cosiddetto *ciclo di vita* di un documento di specifica, indispensabile dato il periodico aggiornamento delle specifiche. Tale ciclo di vita prevede che ogni documento prodotto sia inizialmente in stato *preliminare*, cioè ancora non pronto per essere utilizzato per fini pratici dell'implementatore, poiché non ancora controllato con rigore. In seguito, se si ritiene che le specifiche trattate nel documento siano pronte per passare in fase operativa, tale documento passa allo stato di *experimental*. Se le implementazioni sperimentali basate sulle specifiche di questo documento dimostrano di essere mature per operare anche in ambiti reali di produzione, il documento passa allo stato *standard*. In qualunque di queste fasi il documento può essere sostituito da uno più aggiornato: in questo caso passarà allo stato di *obsolete*.

Può essere utile precisare che al momento della scrittura i documenti facenti parte delle specifiche precedenti FIPA2000 sono considerati obsoleti, mentre le specifiche FIPA2000 sono ancora in fase sperimentale, e diversi documenti sono addirittura allo stato preliminare.

Nel seguito si tratterà dell'architettura astratta, del sistema di gestione (agent managent system) e del supporto per la mobilità, del linguaggio ACL e dei content language utilizzabili con FIPA, del servizio trasporto messaggi e della possibilità di interazione di agenti FIPA con software non basato sul modello ad agenti. Si ritiene che la conoscenza di tali argomenti sia necessaria e sufficiente per la realizzazione di un MAS a norme FIPA. I documenti non ancora presenti al momento della scrittura riguardano i protocolli di comunicazione e i communicative acts: sono pertanto necessari allo sviluppatore di agenti e non al progettista di MAS. È in ogni caso possibile documentarsi su tali argomenti a partire dalle precedenti specifiche.

Non verranno trattati nel dettaglio l'ontology service ed i content language, poiché una loro trattazione adeguata esula dallo scopo che ci si propone.

2.3 – L'architettura astratta

2.3.1 Il perché di una architettura astratta

Si è già detto che la definizione di una *architettura astratta* è una novità delle specifiche FIPA2000. Probabilmente la sua origine è dovuta all'esperienza che comincia ad accumularsi sullo sviluppo di piattaforme per agenti. Le specifiche sono documenti in cui si tratta circa l'implementazione di un

certo sistema (MAS nel caso di FIPA). Le specifiche quindi "dirigono" il processo di progettazione del sistema. Questo si baserà parecchio sulle scelte implementative dettate dal progettista, ma gran parte delle entità del sistema risultano comunque già definite dalle specifiche ad un livello di precisione elevato. Definire una architettura astratta significa invece stabilire un modello estremamente generale e flessibile circa la struttura di un sistema, descrivendo quali siano i concetti di base di funzionamento, detti *elementi* (termine che da ora e per tutto il punto 2.3 verrà usato soltanto in questa accezione), e le *relazioni* tra questi. La contropartita di un'architettura astratta è l'*architettura concreta*, o reale, una delle possibili istanziazioni (o implementazioni, che in questo contesto è uguale) derivabili dall'architettura astratta da cui questa deriva. Far sì che una architettura concreta sia un'implementazione di una architettura astratta implica che almeno i concetti di base, il modello mentale con cui si pensa ad un sistema e alle interazioni che avvengono al suo interno, coincidano con quelli di un'eventuale altra piattaforma. Astrarre quindi, ancora una volta per cooperare, e qui in maniera più precisa per *interoperare*: parola chiave questa per la missione di organismi come FIPA.

L'architettura astratta FIPA si presenta come un'ulteriore astrazione rispetto alle specifiche FIPA stesse, e queste vengono fatte discendere dalla prima come naturale processo deduttivo-implementativo, cioè come costruzione di un concetto a partire da un concetto più astratto.

Come si è detto dal principio, la cooperazione ha senso se occore integrare diversi sistemi per ottenere un sistema unico. L'architettura astratta viene creata dal momento che la varietà di sistemi (a tutti i livelli) oggi è notevole, e che ogni sistema ha pregi e difetti, forze e debolezze. Tutto ciò implica che per ogni sistema esistono modi e situazioni di utilizzo diversi dagli altri. Pertanto, se vogliamo *flessibilità*, cioè la capacità con vari sistemi di adattarsi a varie situazioni, e *interoperabilità*, cioè la capacità dei sistemi di cui sopra di convivere, allora dobbiamo *astrarre*. Ma questo processo di astrazione comporta anch'esso dei limiti: i progettisti FIPA nel corso dello sviluppo hanno incontrato due principali difficoltà, e l'architettura astratta frutto del loro lavoro è il risultato di vari compromessi:

- *una certa area non può essere descritta in modo astratto*: questo succede perché astrarre, in questo caso, vuol dire trovare i punti comuni tra i sistemi disponibili in una certa area, e può succedere che tali punti non siano in numero tale da giustificare una nuova astrazione (a meno che non si voglia creare una astrazione inutile);
- *fino ad ora non è stato possibile descrivere una certa area in modo astratto*, tipicamente perché non si è trovato un punto d'accordo sul come standardizzare l'area in questione, cioè su quali concetti base prendere come fondamenta per la costruzione delle implementazioni in quell'area.

Proprio a causa di questi compromessi l'architettura astratta FIPA non copre tutte le aree che devono essere implementate in un MAS. Un'architettura concreta è pertanto conforme all'architettura astratta FIPA se implementa come *insieme minimo* le caratteristiche di quest'ultima. È possibile verificare questa condizione, dal momento che deve essere possibile mappare ogni componente dell'architettura astratta in un componente dell'architettura concreta. Va precisato che tale funzione di mappatura non è iniettiva, poiché più componenti dell'architettura astratta possono essere realizzati da un solo componente reale. Da questa definizione segue anche che non è solo possibile, ma praticamente sicuro, che una architettura reale abbia componenti aggiuntivi rispetto a quella astratta dalla quale discende. Ed è da questa definizione che si può notare come sia garantita in questo modo notevole flessibilità al progettista di MAS, essendo questi libero di scegliere un

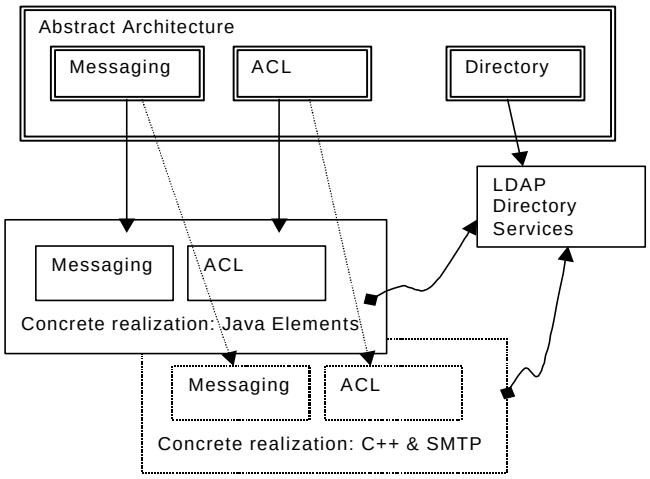


Fig.5 [13]: due implementazioni che condividono un elemento realizzativo

qualsiasi sistema tra quelli disponibili per "concretizzare" una certa area del MAS, indipendentemente da come sono implementate le altre aree. Possono dunque esistere architetture reali diverse che hanno alcune parti in comune, come schematizzato nella fig.5, nella quale due architetture per il resto diverse hanno in comune il servizio directory (del quale parleremo nel seguito), realizzato in entrambe attraverso il sistema LDAP.

L'architettura astratta definisce due tipi di entità di base: gli elementi e le relazioni tra essi. É importante definire alcuni concetti che servono a precisare quali siano le proprietà di queste entità. Chiariamo subito. Un elemento può essere:

- *mandatory* o *optional*: può essere obligatorio, nel senso che una architettura reale conforme a quella astratta deve obbligatoriamente avere un qualcosa che implementa l'elemento astratto considerato, oppure opzionale, ed in questo caso l'architettura concreta non è obbligata ad una sua implementazione;
- *actual* o *explenatory*: actual indica che l'elemento considerato è pensato per potere esistere realmente. Actual è contrapposto infatti ad explenatory, che indica invece un elemento che non è pensato per esistere in una architettura reale, tanto piuttosto per scopi di modellazione del sistema o di chiarificazione. Per esempio un elemento astratto può consistere in un raggruppamento di altri elementi astratti, per cui questi ultimi saranno actual, mentre il primo potrebbe essere explenatory;
- *single* o *functional*: l'essere single implica per un elemento astratto il dover essere implementato da un elemento preciso ed apposito dell'architettura reale; al contrario un elemento functional può essere implementato anche in modo che le sue funzionalità siano divise tra diversi elementi reali.

Per ogni elemento si definisce una terna di valori che specifica se esso è mandatory od optional, actual o explenatory, single o functional.

Una relazione tra elementi può essere anch'essa *mandatory* o *optional*, con lo stesso significato valido per gli elementi.

Il creare un'implementazione concreta di un elemento dell'architettura astratta viene anche detto *realizzare*, e tale implementazione è chiamata *realizzazione*.

Prima di procedere con l'analisi del modello dell'architettura, bisogna chiarire il concetto di *coppia chiave-valore*. É una coppia ordinata di due elementi, solitamente stringhe di caratteri: la chiave, che è un termine facente parte di un dominio amministrato da un ente come (in questo caso) FIPA, ed il valore ad essa associato. Questo è un concetto importante dal punto di vista della flessibilità, perché permette l'introduzione di nuove caratteristiche nell'architettura (introducendo appunto nuovi termini nel dominio delle chiavi) senza entrare in conflitto con le caratteristiche precedenti (visto che i vecchi termini possono continuare ad esistere). Un insieme di coppie chiave-valore è chiamato *key-value tuple* (t-upla di coppie chiave-valore), da cui l'acronimo *KVT*.

2.3.2 Composizione dell'architettura: agenti e servizio directory

Il primo elemento cui conviene far cenno è proprio *agent*, che rappresenta il concetto per cui nasce l'architettura astratta. In realtà questa prevede un elemento più astratto rispetto all'agente, chiamato *FIPA-entity*, che rappresenta il concetto di entità fondamentale. Un agente è una entità fondamentale poiché un MAS viene creato appunto per permettere agli agenti di funzionare, e per il momento è anche l'unica. Ma prima o poi potrebbe esserci il bisogno di far gestire ad un MAS altre entità fondamentali, per cui avere un concetto di entità fondamentale separato da quello di agente potrebbe, in futuro, risultare utile. Da quanto detto segue che FIPA-entity è un elemento explenatory, e per di più optional, cosicchè l'implementatore può benissimo farne a meno e basarsi solo sull'elemento agent. Questo è invece, ed ovviamente, di tipo mandatory / optional / single.
L'elemento *FIPA-service* rappresenta il generico (infatti è di tipo explenatory) servizio di supporto per gli agenti: a questa categoria appartengono *directory service* e *message-transport-service*. L'elemento *action-status* identifica il valore di ritorno che il servizio restituisce all'agente che gli richiede un'operazione: solitamente indica il successo o il fallimento dell'operazione, cioè lo stato dell'operazione. L'elemento *explanation* serve a riportare, se necessario, il motivo di tale stato. L'architettura astratta non specifica quale debba essere la tipologia di implementazione di un FIPA-service: questo può infatti essere implementato come una API (Application Programming Interface) resa disponibile dalla piattaforma e che gli agenti clienti del servizio possono chiamare dall'interno, oppure come un agente separato con cui gli agenti clienti comunicano come con tutti gli altri agenti. Lo stesso vale per il ritorno del risultato delle azioni. Il problema di cosa sia il action-status è lasciato all'implementatore: potrebbe essere il valore di ritorno di una funzione API, un messaggio vero e proprio (nel caso il servizio sia implementato come agente), oppure ancora un oggetto. Le funzionalità dei due servizi definiti da FIPA saranno chiarite una volta compresi i meccanismi fondamentali del modello dell'architettura astratta.
Innanzitutto è previsto che i meccanismi di trasporto dei messaggi (di cui al punto 1.1.3) possano essere più d'uno all'interno di una stessa piattaforma, e questo vuol dire che un agente può essere raggiunto con diversi indirizzi, dal momento che ogni sistema di trasporto ha un suo modo di indirizzamento e localizzazione. Per esempio, un apposito agente può essere raggiunto indifferentemente attraverso l'SMTP, quindi per posta elettronica con un indirizzo del tipo agent@foo.com, con una chiamata ad un ORB, con un indirizzo del tipo iiop://foo.com/agent, o attraverso un sistema di "code messaggi", come un device FIFO di Linux.
Il *transport-description* è l'elemento che identifica un agente su un preciso sistema di trasporto. É composto da:

- *transport-type*: elemento che identifica un certo sistema di trasporto;
- *trasport-specific-address*: elemento che identifica l'indirizzo dell'agente nel sistema di trasporto specificato al punto precedente;
- zero o più *transport-specific-property*: quest'ultimo è un elemento che definisce una certa proprietà associata con un sistema di trasporto. L'insieme di queste proprietà, se presente, aiuta il processo di connessione tra due agenti.

Tutti questi elementi sono mandatory / actual / single, ad eccezione del transport-specific-property che è optional.

Il *locator* è l'elemento che rappresenta tutti i possibili modi di raggiungere un agente (anche se sarebbe più preciso dire un FIPA-entity): è quindi l'insieme di tutti i transport-description di un agente.

A parte il locator, ogni FIPA-entity deve avere un nome universalmente univoco, o più precisamente un GUID (globally univocal identifier) che possa identificarlo non solo all'interno di una piattaforma, ma anche all'esterno. L'elemento che identifica il nome di un FIPA-entity è il *FIPA-entity-name*. In base al cosiddetto modello credenze / desideri / intenzioni, il nome dell'agente può diventare un modo per esprimere ciò che l'agente stesso è, vuole e fa: una simile scelta, pur essendo conveniente, non costituisce un obbligo. Perlopiù, inoltre, allo scopo di generare nomi univoci a livello globale si ricorre a tecniche che fanno uso anche di una generazione di numeri casuali su un dominio molto ampio, in modo da ridurre la possibilità di collisione. È chiaramente importante ai fini dell'interoperabilità che il FIPA-entity-name sia davvero universalmente univoco, e che permanga durante tutto il corso della vita dell'agente senza cambiamenti.

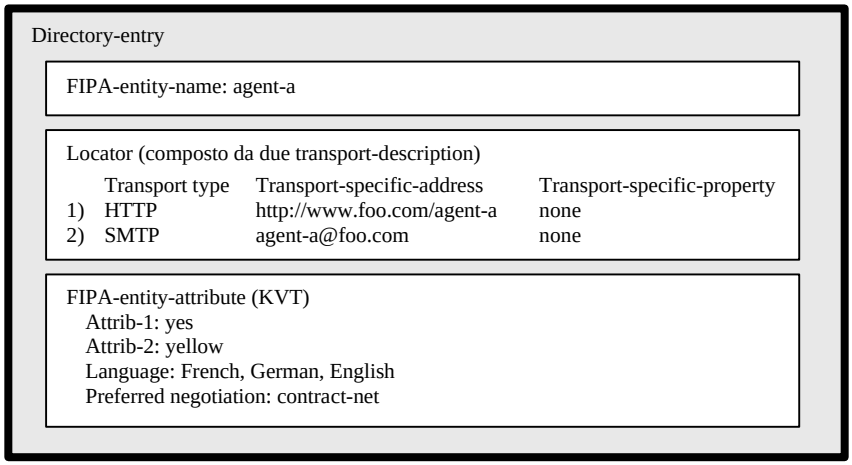


Fig 6: il directory-entry di un ipotetico agente reperibile attraverso due sistemi di trasporto

Un FIPA-entity può avere anche alcuni attributi: tipo dei servizi offerti, costo e restrizioni nell'uso, ed altro. L'elemento che rappresenta un attributo è il *FIPA-entity-attribute*, e viene espresso sotto forma di coppia chiave-valore.

L'insieme degli attributi di un agente è quindi un KVT.

Locator e FIPA-entity-name sono elementi mandatory / actual / single, FIPA-entity-attribute è optional.

Gli elementi principali che l'architettura astratta vede di un dato agente sono quindi:

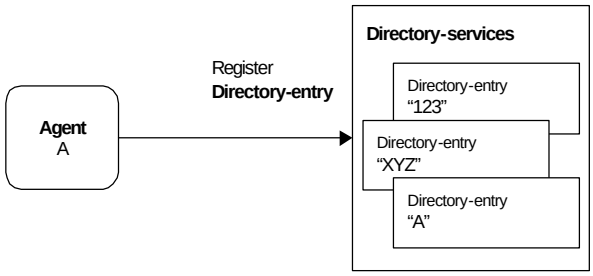


Fig 7 [13]: l'agente A si registra presso il directory-service

- il FIPA-entity-name
- il locator
- zero o più FIPA-entity-attribute

Questi sono gli elementi che compongono il *directory-entry* (mandatory / actual, esempio in fig. 6).
Un agente appena creato registra il suo directory-entry presso il directory-service, nominato in precedenza. Il directory-service è il servizio che permette ad un agente di registrare "la sua esistenza" (una sorta di anagrafe, fig 7), affinchè altri agenti possano localizzarlo e comunicare con lui. Un agente alla ricerca di un altro che svolga per lui un certo compito, per esempio, domanderà (manderà una query, fig. 8) al directory-service quali sono gli agenti da lui registrati che svolgono un certo servizio, con determinati parametri. Il directory-service troverà tutti i directory-entry che corrispondono alla ricerca specificata e li manderà all'agente richiedente, il quale riceverà tali directory-entry, ne sceglierà uno, e da questo trarrà tutto il necessario (GUID, locator) per stabilire una comunicazione con l'agente che per lui svolgerà il compito voluto.
Va notato il fatto che possono coesistere più istanze di directory-service all'interno di una piattaforma, e che ogni tipo di directory-service può disporre di meccanismi per limitare la visibilità dei directory-entry. È possibile richiedere al directory-service, che è un elemento di tipo mandatory / actual, le seguenti azioni, che esso deve obbligatoriamente accettare:

- *register* (registrazione): un agente vuole registrarsi presso il directory-service, invia dunque il suo directory-entry ad una delle istanze di directory-service presenti nel sistema, se occorre con alcuni parametri di visibilità. Quest'ultima non viene trattata dall'architettura astratta, e pertanto può essere resa in modi differenti all'interno di ciascuna realizzazione. Se la registrazione viene effettuata correttamente, il servizio riporta un action-status indicante successo, altrimenti riporta un action-status indicante fallimento, assieme con un explanation, specificante il motivo del

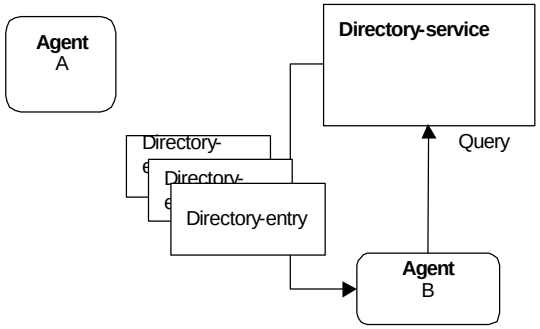


Fig 8 [13]: un agente B fa una query al directory-service

fallimento: le possibilità, in questo caso, sebbene dipendano dall'implementazione, in linea di massima sono tre: 1) *duplicate*: il nuovo directory-entry collide con uno già presente nel directory-service, magari perché i due hanno uno stesso FIPA-entity-name; 2) *access*: l'agente non ha il permesso di richiedere l'operazione; 3) *invalid*: il directory-entry fornito non è valido.

- *modify* (modifica): un agente vuole cambiare le sue impostazioni di registrazione, ed invia un nuovo directory-entry ad una delle istanze di directory-service, con eventuali parametri di visibilità. Il servizio confronta le coppie chiave-valore del nuovo directory-entry con quelle del precedente, il quale sarà aggiornato in maniera opportuna: nuove chiavi vengono inserite assieme al relativo valore, chiavi presenti sia nel vecchio che nel nuovo directory-entry vedono aggiornato il loro valore, chiavi non presenti nel nuovo directory-entry rimangono inalterate. Una chiave può essere eliminata lasciando nel nuovo directory-entry il relativo valore in uno stato di "non impostato" (per esempio stringa vuota). Il servizio comunica lo stato dell'operazione attraverso l'action-status, allo stesso modo dell'operazione di registrazione. In questo caso i probabili motivi di fallimento sono: 1) *not-found*: se si richiede di modificare un directory-entry non presente; 2) access: come per l'operazione di registrazione; 3) invalid: come per l'operazione di registrazione.
- *delete* (cancellazione): un agente vuole cancellare il suo directory-entry dal servizio. Per far ciò deve inviare al directory-service un directory-entry che contenga il suo FIPA-entity-name (il resto del directory-entry è insignificante). Il servizio comunica lo stato dell'operazione attraverso l'action-status, allo stesso modo dell'operazione di registrazione. In questo caso i probabili motivi di fallimento sono gli stessi dell'operazione di modifica.
- *query* (interrogazione): un agente chiede quali siano i directory-entry registrati che corrispondono ad un directory-entry fornito. La ricerca viene effettuata sulle coppie chiave-valore: per ogni chiave presente nel directory-entry fornito deve essere presente la stessa chiave nel directory-entry che si sta controllando, ed il relativo valore deve essere uguale tra i due. Non ha importanza quindi se un directory-entry esistente contiene altre chiavi oltre quelle specificate in quello fornito. Il servizio comunica lo stato dell'operazione attraverso l'action-status, allo stesso modo dell'operazione di registrazione. In questo caso i probabili motivi di fallimento sono: 1) not-found: nessun directory-entry presente corrisponde a quello fornito; 2) access: come per l'operazione di registrazione; 3) invalid: come per l'operazione di registrazione. A questi tre casi si aggiunge l'eventualità che il directory-entry fornito non sia valido.

Probabilmente la piattaforma si estenderà su più host, o su parecchie reti, per cui è indispensabile implementare il directory-service attraverso un sistema che permetta la gestione di directory in ambiente distribuito: tra questi troviamo per esempio X.500 e la sua versione ridotta, LDAP (Lightweight Directory Access Protocol), NIS, Microsoft Active Directory, Novell NDS. Va qui precisata la differenza tra il directory-service dell'architettura astratta ed il Directory Facilitator (DF) delle normali specifiche FIPA. Il primo è un servizio astratto, nel senso che è il massimo che i progettisti FIPA sono riusciti a standardizzare in quest'area, e permette di compiere sui directory-entry ricerche su singolo livello. Ricerche più complesse, su molteplici livelli, sono invece richieste dal DF, che non può essere implementato semplicemente con servizi di directory distribuiti come quelli visti in precedenza, ma richiede anche l'uso di sistemi che permettano livelli di innesto non limitati, come LISP o Prolog. E questo è un problema che deve risolvere totalmente il progettista del MAS.

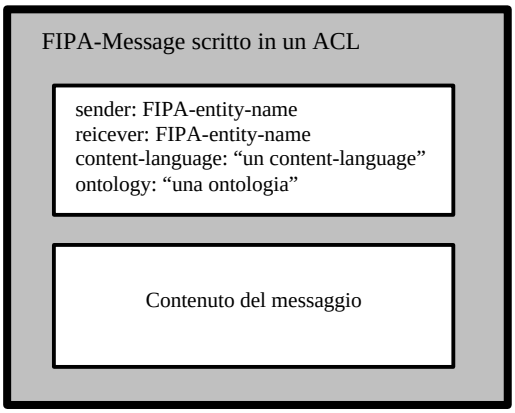


Fig. 9: schema generale di un FIPA-message

2.3.3 Composizione dell’architettura: messaggi e servizio trasporto

Gli agenti sono entità che colloquiano scambiandosi messaggi. Ora si vedrà come l’architettura astratta modellizza questa situazione. Elemento base è qui il *FIPA-message*, che , come indicato dal nome, si riferisce al singolo messaggio inviato da un agente ad un altro. Il *agent-communication-language* rappresenta il linguaggio usato per scrivere il messaggio: un esempio ne è il FIPA ACL. L’architettura astratta distingue tra la parte di messaggio indicante il vero contenuto (che da adesso chiameremo semplicemente contenuto), cioè quella che esprime conoscenza, rispetto alla parte “di servizio”, che contiene tutte le informazioni necessarie affinchè il messaggio venga compreso (il concetto sarà chiarito tra poco). A questo proposito l’architettura astratta definisce un elemento *content*, che identifica il contenuto di un FIPA-message, ed un elemento *content-language*, che identifica il linguaggio usato per esprimerlo. FIPA stabilisce con esattezza quali possano essere i

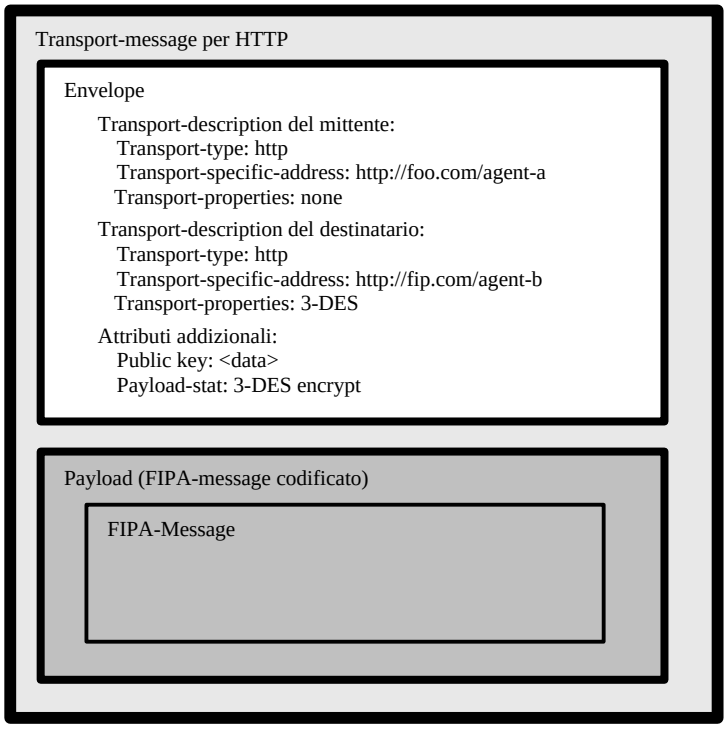


Fig 10: esempio di transport-message: dopo la codifica triplo-DES a chiave simmetrica del payload, sarà inviato per mezzo del protocollo HTTP

linguaggi per il contenuto presenti in una piattaforma conforme alle sue specifiche (vedere 2.4.3).
Due elementi importanti della parte "servizio" di un messaggio sono sicuramente il mittente ed il destinatario, espressi questi in termini di FIPA-entity-name, e non di transport-specific-address (cioè identificano, ma non localizzano). Altri due elementi altrettanto importanti sono l'identificazione del content-language usato per rappresentare la conoscenza, ed il riferimento all'ontologia, cui va fatto ricorso per la comprensione dei termini usati nel messaggio. L'ontologia, semplificando, lega i termini di una certa area della conoscenza al loro significato semantico, quindi dà modo al mittente del messaggio di accordarsi col destinatario sull'uso dei termini compresi nel messaggio stesso. L'elemento che rappresenta il concetto di ontologia è *ontology*. L'architettura astratta comunque tratta questo argomento solo a livello di definizione, dal momento che FIPA2000 prevede un documento apposito sull'argomento delle ontologie (vedi fig.4).
La struttura di un FIPA-message è rappresentata informalmente in fig. 9.
Inviare un messaggio su un certo canale di trasporto implica però conoscere l'indirizzo del destinatario, per quel preciso canale. Tale indirizzo il mittente può trovarlo nel directory-entry ottenuto da una query al directory-service (come descritto in 2.3.2). Il modello prevede che, a questo punto, il FIPA-message venga incapsulato in un *transport-message*. Questo elemento identifica il vero e proprio insieme di dati inviato per la comunicazione. Prima di essere incapsulato, il FIPA-message può essere codificato in vari modi attraverso algoritmi di compressione, per aumentare l'efficienza, e/o di crittografia, per aumentare la sicurezza. Anche il transport-message sarà quindi diviso in parte di servizio e parte utile:

- *envelope* è l'elemento che identifica la parte di servizio, e contiene i transport-description del mittente e del destinatario, e le informazioni necessarie per la decodifica del FIPA-message. L'elemento che identifica queste informazioni è chiamato *message-encoding-representation*.
- *payload* invece è l'elemento che identifica il FIPA-message codificato.

Un esempio di transport-message è riportato in fig 10.
Va chiarito che il message-encoding-representation non identifica solamente la codifica del payload, ma anche la rappresentazione (con il significato preciso visto in 1.2.3) del FIPA-message: questo per esempio può essere composto solo da una semplice stringa di caratteri, oppure può essere una stringa impacchettata in un oggetto String o StringBuffer di Java, oppure ancora può essere rappresentato in XML.
Tutti gli elementi fin qui visti sono mandatory / actual / single con l'eccezione di agent-communication-language, message-encoding-representation e ontology, che sono functional.
La fig. 11 visualizza il processo di invio di un messaggio.
È chiaro, da quanto finora detto, che il concetto di meccanismo di trasporto è fondamentale. L'architettura astratta definisce infatti l'elemento *transport* (mandatory / actual) per riferirvisi.
Avendo un transport-message pronto, un agente deve ora veicolarlo sul transport cui è destinato. Il progettista di architetture concrete o di specifiche è libero a questo punto di scegliere tra due possibilità: fare gestire agli agenti i dettagli dell'invio dei transport-message su ogni transport desiderato, oppure ricorrere all'uso del message-transport-service, cioè di un servizio che gestisca per conto dell'agente tutte le problematiche relative all'uso dei transport. La seconda possibilità implica che sia il MAS ad occuparsi della gestione dei transport, e quindi che dipenda solo dal MAS il numero dei transport supportati. Da quanto detto segue che il message-transport-service, che chiameremo anche "servizio trasporto", è un elemento optional, oltre ad essere actual / functional. Se

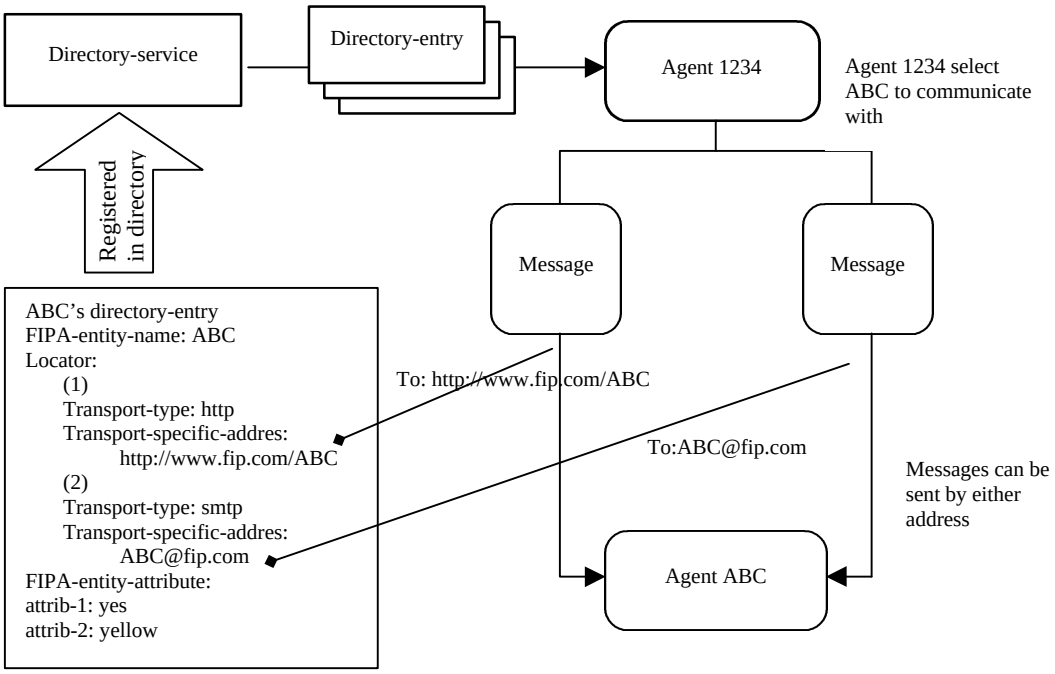


Fig. 11 [13]: l'agente 1234 può raggiungere ABC attraverso due diversi transport

si prevede che una piattaforma farà uso di un solo meccanismo di trasporto, sarà ovviamente conveniente non implementare un message-transport-service, ma lasciare il compito dell'invio alle normali API disponibili a livello agente. Anche se si è detto che l'architettura astratta non specifica in quale modo implementare i servizi per gli agenti, è ovvio che non è possibile implementare il message-tranport-service come agente (bisognerebbe inviargli un messaggio, e per farlo bisognerebbe avere un altro message-transport-service), dunque questo farà parte della struttura che compone la base della piattaforma -in generale: sarà posizionato ad un livello sottostante agli agenti, nella stratificazione software-. Segue l'elenco di azioni che il servizio trasporto può svolgere:

- *bind transport* (connessione): può accadere che, se un agente pensa di intrattenere un lungo dialogo con un altro, ci sia bisogno di creare una connessione (circuito virtuale) tra i due punti. Questo ovviamente ha senso se il sistema sottostante di trasporto supporta le connessioni (p.es. TCP o UDP). In questo caso l'agente che vuole iniziare il dialogo chiede la creazione di una connessione passando al servizio trasporto un transport-description. Questo può anche essere incompleto in più parti, nel qual caso il message-transport-service cercherà di riempire le parti mancanti. Nel caso non si incontrino problemi nella creazione della connessione, il servizio trasporto ritornerà un action-status indicante successo ed un transport-description completo in ogni sua parte, e da quel momento, fin quando possibile, i messaggi scambiati dai due agenti saranno inviati su quella connessione. Altrimenti all'agente sarà ritornato un action-status indicante fallimento insieme con un explanation. Il documento dell'architettura astratta fa riferimento a due probabili casi di errore: 1) access: l'agente non è autorizzato a creare la connessione; 2) invalid: il transport-description fornito al servizio trasporto non è valido. Verosimilmente, a questi si aggiunge l'eventualità che non sia possibile creare la connessione per problemi del sistema di trasporto sottostante (leggi problemi di rete).
- *unbind transport* (disconnessione): non avendo più messaggi da scambiare col suo interlocutore, un agente chiede che la connessione venga chiusa. L'agente passa al servizio trasporto il

transport-description che si riferisce alla connessione da chiudere. Nel caso la richiesta possa essere accolta, il servizio trasporto rilascerà tutte le risorse correlate a tale connessione, e ritornerà all'agente un action-status indicante successo. Altrimenti il servizio ritornerà un action-status indicante fallimento insieme con un explanation. I probabili motivi di fallimento sono: 1) not-found: se si chiede di terminare una connessione inesistente; 2) access: come per l'operazione di bind; 3) invalid: come per l'operazione di bind.

- *send message* (invio messaggio): un agente vuole inviare un messaggio ad un destinatario per il quale non è stata precedentemente creata una connessione. L'agente passa al servizio trasporto il transport-message (non il transport-description). Ricevuto il transport-message il servizio trasporto si incarica, prima, di analizzarne l'envelope per decidere come gestirlo e, dopo, di inviarlo sul canale di trasporto. È importante notare che l'action-status ritornato all'agente non si riferisce al fatto che il messaggio sia giunto oppure no a destinazione, bensì al fatto che la fase di analisi dell'envelope non abbia riscontrato problemi. Non è previsto insomma che l'agente possa sapere se il messaggio è stato correttamente ricevuto. L'explanation di ritorno indicherà il perchè del fallimento dell'analisi dell'envelope. Casi probabili sono: 1) access: come per l'operazione di bind ; 2) invalid: come per l'operazione di bind.
- *deliver message* (consegna messaggio): un agente vuole inviare un messaggio ad un destinatario con il quale ha avviato precedentemente una connessione. Si fa riferimento a quanto detto per l'operazione bind transport.

2.4 L'Agent Communication Language

2.4.1 FIPA ACL: ontologie FIPA

Il FIPA ACL è uno dei possibili agent-communication-language, cioè un linguaggio che gli agenti possono usare per scambiarsi messaggi con sintassi semplice, ma ricchi dal punto di vista semantico. Ovviamente il termine "messaggio" corrisponde al concetto di FIPA-message dell'architettura astratta. Grazie all'uso delle ontologie è possibile, con linguaggi di questo genere, esprimere conoscenza.

Ci si sofferma qui un istante sul discorso delle ontologie. L'ontologia è definita come un insieme di termini con una interpretazione (semantica) associata che possono essere condivisi da una comunità di agenti. Ciò che offre l'ontologia in realtà è un insieme di simboli che rappresentano tutte le possibili proprietà e relazioni tra i concetti esistenti nel dominio dell'ontologia considerata (sull'argomento cui l'ontologia è dedicata). I termini vengono derivati, attraverso composizione logica, a partire dalle suddette proprietà e relazioni, e da altri termini. È proprio questa composizione a conferire una semantica al termine. Ed è proprio per questo motivo che un messaggio deve fare riferimento ad una o più ontologie, in modo tale che il mittente possa utilizzare i termini di queste nel contenuto, e condividere la semantica con il destinatario. Si avrà la certezza che i termini usati

nel messaggio saranno interpretati (con il significato preciso di 1.2.3) alla stessa maniera, cioè faranno riferimento allo stesso concetto.

Al punto 2.5.1 si discuterà degli elementi principali costituenti lo scheletro delle piattaforme FIPA, e di come questi elementi derivino dall'architettura astratta. Brevemente diremo qui che due di questi componenti (vedi 2.1), l'Agent Management System (AMS) ed il Directory Facilitator (DF), permettono rispettivamente la gestione del ciclo di vita degli agenti, quindi i rapporti tra questi e la sottostante piattaforma, e la registrazione dei servizi resi disponibili da ciascun agente, in modo da poter rendere pubblicamente accessibile un elenco dei servizi ed agenti correlati[2]. Si è introdotto l'argomento poiché le specifiche impongono che gli agenti comunichino con l'AMS ed il DF attraverso l'ACL, per cui vedono questi elementi come fossero altri agenti[3]. Se è prevista una comunicazione attraverso ACL bisogna ricorrere all'uso di ontologie adatte, che riguardino i concetti base ed architetturali delle piattaforme (per l'AMS) e lo scambio di informazioni sui servizi resi disponibili da ogni componente (per il DF). Tali ontologie sono create e descritte a priori da FIPA, poiché su di esse si baserà lo scambio di informazioni vitali per il funzionamento della piattaforma. Ci riferiremo a queste ontologie, e solo ad esse, con il termine "ontologie FIPA". Nel corso dei punti successivi si cercherà di illustrarle.

Ma per offrire un buon legame con i documenti originali di specifica si preciserà come in questi si descrivono le ontologie FIPA. In ogni ontologia FIPA si definiscono i *frame* che le appartengono. Un frame non è altro che il simbolo (nel nostro caso un termine) con cui si è deciso di rappresentare un certo concetto, compreso nel dominio dell'ontologia considerata. Per esempio l'ontologia FIPA-Agent-Management (verrà discussa in seguito) comprende il concetto di "identificatore di agente", ed a questo concetto associa il simbolo *agent-identifier* che per l'appunto è un frame di questa ontologia. Di questo frame va espressa la semantica, cioè va definito il concetto ad esso relativo. Questo viene fatto nella maniera più precisa possibile, ma senza usare linguaggi semantici formali: in poche parole ne si dà una descrizione scritta in lingaggio naturale, una *description*. Un frame però può contenere uno o più *parameter* (parametri), che rappresentano, per così dire, "sottoconcetti", ovverossia concetti di base di cui è composto quello principale rappresentato dal frame. Un parametro di agent-identifier è, per esempio, *name*: si è ritenuto, a rigor di logica, che un identificatore di agente dovesse contenere il suo nome. Un altro parametro di agent-identifier è *address*: si è ritenuto che l'identificatore dovesse contenere l'indirizzo dove trovare l'agente. Di ogni parametro va espressa la semantica, come per il frame. Bisogna indicare se questi è obbligatorio (mandatory) per l'esistenza di un frame, oppure no (optional), bisogna indicarne cioè lo stato di *presence*: per esempio, ha senso un agent-identifier non contenente l'address, poiché il nome di un agente è già un GUID, quindi in grado di identificare univocamente un agente. La presenza del parametro address è perciò opzionale. Bisogna indicare quale tipo di informazione può rappresentare i possibili valori del parametro, il *type* del parametro: l'address per esempio può essere espresso come un URL di rete, quindi una stringa con una precisa sintassi; il nome può essere espresso da una stringa senza formato; altri parametri potrebbero essere espressi come numeri interi binari o BCD, come valori di data/ora, come numeri floating-point, ma anche come insiemi o sequenze di più di questi. Infine, per ogni parametro bisogna indicarne i *reserved values* (valori riservati): all'interno

[2] È chiara la correlazione tra il Directory Facilitator ed il directory-service dell'architettura astratta

[3] Può essere che siano implementati come agenti, ma le specifiche FIPA non obbligano strettamente che l'implementatore proceda in questo modo. Maggiori chiarimenti saranno dati in 2.5.1.

del dominio di un parametro, alcuni valori possono essere riservati, tali cioè da essere già assegnati a priori da FIPA, similmente in tutto e per tutto alle parole riservate dei linguaggi di programmazione. Esempio: gli agenti possono registrarsi presso il DF per rendere pubblici i loro servizi; ma è anche possibile che un DF si registri presso un altro DF, costituendo una federazione di DF; per il servizio svolto dal DF, FIPA ha riservato una stringa apposita: *fipa-df*; nessun altro agente potrà pubblicizzare con questo nome un suo servizio.
Oltre ai frame, le ontologie FIPA definiscono anche delle *function*, che corrispondono a quelle che nel linguaggio naturale sono le azioni. Di una function bisogna indicare: quali sono gli agenti che la supportano, attraverso l'indicazione *supported by*; una descrizione in linguaggio informale (così come per frame e parametri), attraverso l'indicazione *description*; il *domain*, cioè il dominio della funzione o, che dir si voglia, i frame sui quali la funzione agisce; l'*arity*, che indica il numero di argomenti della funzione, il cui dominio è infatti una n-upla ordinata di frame; il *range*, cioè il tipo, o frame, del valore di ritorno, ammesso che ve ne sia uno: il risultato della funzione apparterrà al tipo specificato in range. Si dirà inoltre che il valore di ritorno sarà un frame, e che verrà comunicato al richiedente l'azione attraverso un messaggio di ritorno.
Per schematizzare, si riscrivono gli elementi usati per definire i concetti nelle ontologie FIPA…:

- frame: simbolo impiegato per rappresentare il singolo concetto;
- parameter: un frame può essere composto (solitamente lo è) da più parametri;
- description: descrizione in linguaggio naturale, ma il più possibile precisa, della semantica di un frame o, se il frame è composto da più parametri, della semantica di ognuno di questi;
- presence (associato al parametro): indica se la presenza di un parametro è obbligatoria o meno per la descrizione del frame;
- type (associato al parametro): il dominio cui appartengono i valori dei parametri;
- reserved values (associato al parametro): valori riservati nel dominio dei valori dei parametri;

… e gli elementi usati per definire le azioni nelle ontologie FIPA:

- function: simbolo impiegato per rappresentare un'azione;
- supported by: gli agenti capaci di svolgere l'azione;
- description: descrizione in linguaggio naturale ma il più possibile precisa, della semantica dell'azione;
- domain: gli oggetti dell'azione, gli argomenti della funzione;
- arity: numero degli argomenti;
- range: tipo del risultato della funzione.

2.4.2 FIPA ACL: introduzione e struttura

Nel corso della descrizione dell'architettura astratta si è già visto quali sono gli elementi salienti di un messaggio di un generico ACL. Si metterà qui l'accento sugli elementi e sulla struttura dei messaggi nel FIPA ACL. Il FIPA ACL è il linguaggio standard di FIPA per le comunicazioni tra agenti, e d'ora in avanti verrà indicato semplicemente con il termine ACL, a meno che non diversamente specificato. L'ACL è costituito da un insieme di campi, per ognuno dei quali è definito un nome. Bisogna porre l'accento sul fatto che, qui di seguito, si parlerà di questi campi e del loro contenuto, riferendosi però sempre ad una "sintassi astratta": in una implementazione reale i campi

saranno codificati, o meglio rappresentati, in qualche forma. È possibile che in una implementazione si scelga di usare una rappresentazione sotto forma di semplici stringhe, per cui nel messaggio appariranno effettivamente le stringhe con i nomi dei campi. Ma è anche possibile che una implementazione rappresenti i campi come oggetti, per cui un messaggio sarà costituito da un insieme di tali oggetti, ognuno dei quali rappresenta un campo. Oppure è possibile una rappresentazione numerica in cui sia associato un numero standard ad ogni campo, così come ad ogni suo valore possibile. Nell'architettura astratta si è introdotto apposta il concetto di message-encoding-representation, elemento del transport-message che informa il destinatario sul metodo di codifica, quindi della sintassi usata, dall'ACL. Ovviamente, affinchè due agenti possano scambiarsi messaggi ACL, devono trovare un metodo di rappresentazione comune. FIPA cerca il più possibile di standardizzare questi metodi, ma non è possibile definire precisamente quali debbano o non debbano essere supportati da un agente FIPA conforme. La diversità nelle architetture su cui vengono costruiti gli agenti fa sì che ognuna abbia, per così dire, un modo di rappresentazione preferito, se non altro per la diversa visione del mondo delle suddette architetture. Così un agente Java o C++ si troverà a suo agio con rappresentazioni ACL basate sulla composizione di oggetti quanto un agente PERL con una rappresentazione mediante stringhe, per via della sua gestione estremamente efficace di queste ultime. E limitare l'uno o l'altro, obbligandolo in codifiche non congeniali, vorrebbe dire togliere, in qualche modo, potenzialità. Quello che si è deciso di fare in sede FIPA è stato di definire delle rappresentazioni "consigliate", per far sì che, se non altro, sia altamente probabile trovare un metodo di codifica comune. É possibile che una implementazione definisca suoi propri campi, che non sono comunque richiesti per la conformità alle specifiche FIPA. Si è già detto in precedenza (vedi 1.2.4) che l'ACL si basa sulla teoria degli speech act's: l'unico campo obbligatorio in un messaggio ACL è, per l'appunto, il *performative*, che contiene il comunicative act che contraddistingue il messaggio. Performative costituisce inoltre il primo campo del messaggio, ed è l'unico di cui non compare il nome (gli altri campi sono composti da una coppia nome campo – valore associato). Si riportano di seguito tutti i campi definiti da FIPA:

- performative: contiene il tipo di communicative act che contraddistingue il messaggio;
- *sender*: identifica il mittente del messaggio e può essere omesso se questi vuole rimanere anonimo; si vedrà più avanti come è costituito un identificatore. Nota: il comunicative act è espresso dal punto di vista del mittente: un messaggio con CA "inform" implica che sia il mittente che vuole informare il destinatario di qualcosa, non che il mittente voglia essere informato dal destinatario. Il mittente è colui che svolge l'attività correlata con il CA;
- *receiver*: identifica il destinatario e può essere omesso solo se questi è identificabile altrimenti dal contesto del messaggio, o in casi particolari. È possibile per un messaggio avere più destinatari, in modo da stabilire una comunicazione multicast (uno a molti);
- *reply-to*: indica che la risposta del destinatario non dovrà essere inviata al mittente (campo sender) del messaggio, ma all'agente identificato da questo campo;
- *content*: è il contenuto vero e proprio del messaggio, espresso in un content language; non è un campo obbligatorio perché, in certe occasioni, può capitare che il CA esprima da solo il senso della comunicazione: per esempio il CA cancel riferito ad un messaggio precedente nella comunicazione; il significato dei termini usati nel content è dato dalle ontologie specificate nel campo apposito;

- *language*: specifica qual è il linguaggio formale usato per esprimere il contenuto; può essere omesso perché è possibile che il destinatario sappia già a priori il content language usato nel messaggio, magari perché si è stabilito precedentemente nel corso della conversazione tra i due agenti;
- *encoding*: come si è visto, l'envelope del transport-message include le informazioni message-encoding representation che specifica il metodo di codifica dell'ACL; è però possibile che il content sia espresso con un'ulteriore codifica diversa dalla precedente: questo campo indica la codifica del content (più avanti si darà qualche informazione aggiuntiva);
- *ontology*: specifica le ontologie necessarie per la comprensione dei termini usati all'interno del content; il campo può essere omesso quando si fa riferimento ad ontologie particolarmente comuni, tali che il destinatario possa riconoscerle senza che siano esplicitate;
- *protocol*: specifica il protocollo di interazione da usare nella conversazione tra due agenti; si chiarirà il concetto tra poco;
- *conversation-id*: contiene un identificatore univoco della conversazione: permette di identificare una conversazione tra tutte quelle svoltesi nel tempo da un agente. Può per esempio essere utile per legare tra loro messaggi di conversazioni svoltesi in precedenza e registrate;
- *reply-with*: poichè un agente può intrattenere contemporaneamente più conversazioni diverse, magari con uno stesso interlocutore (nulla vieta la richiesta contemporanea di più servizi dello stesso tipo), l'utilizzo di questo campo può essere utile per correlare tra loro i messaggi di ognuna di tali conversazioni; il campo reply-with (rispondi con…) contiene infatti un identificatore che il destinatario inserirà nel campo in-reply-to (in risposta di…) del messaggio di risposta; la funzione è comunque simile a quella di conversation-id, con la differenza che quest'ultimo presuppone l'identificazione univoca di tutte le conversazioni di un agente;
- *in-reply-to*: vedi reply-with;
- *reply-by*: contiene una espressione di data o di ora: è usato per comunicare il tempo entro cui si spera di ottenere un messaggio di risposta; tale messaggio potrà essere identificato mediante l'uso dei campi conversation-id e/o reply-with/in-reply-to.

Ognuno di questi termini è compreso nell'ontologia "FIPA-ACL", e precisamente come singolo parametro del frame FIPA-ACL-Message. L'ontologia suddetta è prevista appositamente per riferirsi agli elementi di un messaggio ACL.

2.4.3 FIPA ACL: protocolli e content languages

"Il protocollo di interazione definisce una sequenza di messaggi che rappresenta una conversazione completa tra due agenti" [6]. Probabilmente è il modo migliore per esprimere sinteticamente il concetto di protocollo di interazione. FIPA infatti definisce alcuni pattern, o sequenze, di CA da adoperare in sequenza nello scambio di messaggi in una conversazione, in modo da giungere ad una conversazione di senso compiuto. Un esempio ([6]) dovrebbe chiarire il concetto: in fig. 12 è schematizzato lo scambio di messaggi, con relativo CA (il performative, in corsivo) e l'eventuale content (in grassetto, nello schema si esprime solo il concetto generale), che costituisce il protocollo *FIPA-request*. Un agente A può, tramite questo protocollo, richiedere un'azione ad un agente B, e venire informato da questo sullo svolgimento di tale azione.

Non è obbligatorio usare protocolli di interazione per dare una struttura alla conversazione. Tale struttura può infatti decidersi al momento del dialogo, in maniera dinamica. Ma questo aumenta le difficoltà di programmazione di entrambi gli agenti coinvolti in tale dialogo. Le specifiche definiscono "estremamente ambizioso" il progettare un agente che basi il controllo delle conversazioni direttamente sulla semantica dell'ACL piuttosto che sul supporto dei protocolli.

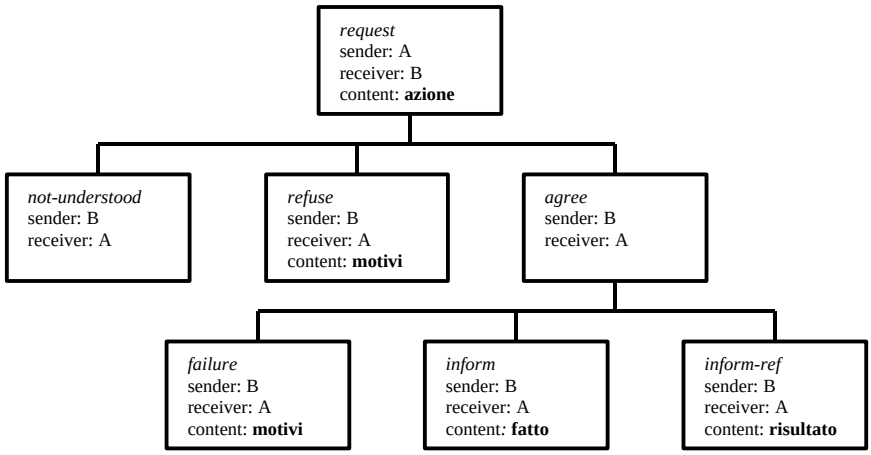


Fig 12: sequenze possibili di messaggi in una conversazione utilizzante il protocollo FIPA-request

Si è detto che il contenuto vero e proprio dei messaggi è espresso nel campo content attraverso un content language. La scelta di quali linguaggi adottare allo scopo è fondamentale, perché riguarderà da vicino la costruzione degli agenti. Questi infatti dovranno incorporare i parser per tutti i linguaggi supportati, ed è quindi importante che la sintassi sia semplice. Le proprietà di un linguaggio usato per esprimere il content influenzeranno notevolmente le capacità espressive di un agente durante una conversazione. É doveroso dunque differenziare: capacità espressive di un linguaggio potrebbero essere molto consone ad alcune situazioni particolari, molto sconvenienti in altre; inoltre, alcuni linguaggi possono essere estremamente generali e flessibili, offrendo costrutti validi in ogni situazione, altri possono essere più settoriali e meno flessibili, offrendo supporti più adatti per esprimere certe categorie di concetti. L'organismo che mantiene l'elenco dei content language che è possibile usare è, ancora una volta, FIPA. Questi definisce la cosiddetta "Content Language Library" (CLL): una serie di documenti in cui si pubblica l'elenco dei linguaggi "ammessi" e le specifiche di ognuno di questi. La CLL di FIPA 2000 ne contiene quattro:

- FIPA SL (Semantic Language)
- FIPA CCL (Constraint Choice Language)
- FIPA KIF (Knowledge Interchange Format)
- W3C RDF (Resource Description Framework)

Ovviamente un agente non deve supportare tutti i linguaggi, ma può scegliere quello più congeniale al settore relativo di applicazione, fermo restando che la scelta avrà ripercussioni sulla interoperabilità: un agente potrà colloquiare solamente con agenti con cui condivide almeno un content language. È comunque fatto obbligo all'agente di disporre dei mezzi necessari per poter scambiare, con l'AMS e tutti gli altri agenti, almeno le informazioni indispensabili per l'agent management. Per queste si provvede solitamente attraverso il FIPA SL, che è possibile considerare

come un content language standard. Per semplificare l'implementazione degli agenti si è deciso di proporre anche versioni ridotte della sintassi normale del linguaggio SL, così che un agente non sia costretto ad implementare un parser completo SL solo per le operazioni di agent management, se per tutto il resto provvede con altri content language. Tali versioni ridotte prendono nome, in ordine di complessità crescente, di: SL0, SL1, SL2.
Per tutti i linguaggi della CLL è prevista una codifica in forma di stringhe, ma come si è detto non è l'unica disponibile.
Si è anche visto che la codifica del content del messaggio va indicata nel campo encoding (vedi campo encoding più sopra in questo paragrafo). Per ogni linguaggio però è prevista una codifica standard, che è implicita in assenza del campo encoding.

2.5 – Il modello di riferimento

2.5.1 Derivazione del modello di riferimento

Come si è detto al punto 2.3.1, l'architettura astratta FIPA presenta un modello estremamente generale di piattaforma ad agenti, quel modello cioè che risulta applicabile pressocchè in ogni situazione, e che, descrivendo relazioni astratte ma espressive della logica di funzionamento del sistema, si presenta come base per lo sviluppo di piattaforme ad agenti interoperabili. Sull'architettura astratta FIPA si poggiano le specifiche FIPA vere e propie. Le specifiche infatti, come si è gia detto, vincolano ulteriormente il progettista di MAS, poiché per l'appunto specificano vari altri parametri che devono essere rispettati affinchè il MAS risulti ad esse conforme. Le specifiche riprendono i concetti dell'architettura astratta e li estendono fino a poter fornire tutte le idee di base per lo sviluppo di una architettura concreta. Ecco quindi che il directory-service diventa il *Directory Facilitator*, il message-transport-service diventa *Message Transport System*, si perde il

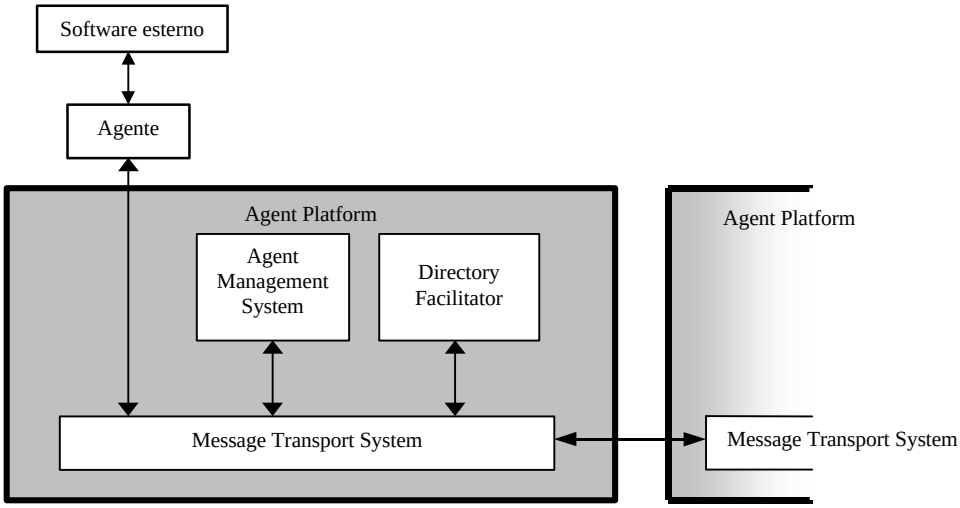


Fig. 13: Schema del modello di riferimento FIPA

concetto astratto di FIPA-entity per concentrarsi solo su quello più preciso (almeno per quanto visto in 1.2.1) di agente, si introduce la gestione del ciclo di vita dell'agente (agent life-cycle) insieme con il nuovo componente ad essa dedicato, l'*Agent Management System*. Viene inoltre specificato un particolare ACL, il FIPA ACL, assieme a tutti i content language ad esso connessi. Questo linguaggio costituirà la base della cooperazione tra agenti sulla stessa piattaforma e tra agenti su piattaforme diverse. Tutti questi elementi (termine usato qui informalmente) vanno a costituire il cosiddetto *modello di riferimento* (reference model) FIPA per lo sviluppo di piattaforme ad agenti.
Nella fig. 13 è visibile uno schema di principio del modello di riferimento, che comprende quattro elementi fondamentali:

- l'agente
- l'Agent Management System, AMS: elemento obbligatorio;
- il Directory Facilitator, DF: è una specializzazione del directory-service dell'architettura astratta;
- il Message Transport System, MTS: è una specializzazione del message-transport-service dell'architettura astratta, ma, a differenza di quest'ultimo, è obbligatorio;

L'agente è l'entità per la quale si sviluppa l'AP; nella figura non è incluso all'interno del blocco rappresentante l'AP perché può essere visto come qualcosa che opera ad un livello superiore, utilizzando le funzionalità previste dall'AP.
Un agente deve avere un suo proprietario (owner), sia esso una persona fisica (un utente) o un'organizzazione o società; deve avere un nome che lo identifichi univocamente a livello globale; deve collegarsi, attraverso il MTS oppure in maniera propria, ad uno o più meccanismi di trasporto: l'architettura astratta prevede anch'essa gli ultimi due requisiti citati. L'agente è libero di accedere a software esterno al sistema considerato, usando il metodo più conveniente, avendo quindi la possibilità di agire come broker (mediatore) tra altri agenti ed il suddetto software: si analizzeranno i modi di interfacciamento con software non "ad agenti" al punto 2.6 . Parlando di un agente si usa il termine HAP (dall'inglese "home agent platform") per riferirsi alla piattaforma su cui è stato creato. Si ricorda che l'agente può essere mobile e quindi può decidere di migrare da una AP ad un'altra, se tutte e due queste ultime supportano la mobilità e lo stesso tipo di codice eseguibile. Può sicuramente essere utile precisare che, quando si parla di un agente, del suo nome, e del suo proprietario, ci si sta riferendo non al software in quanto tale, cioè al programma, ma ad una sua singola istanziazione, cioè al processo in esecuzione, che è dotato di un suo preciso stato interno (cioè del contenuto delle strutture dati) e si trova in un preciso stato esecuzione (vedi 2.5.2).
L'AMS ed il DF costituiscono i punti focali di una architettura concreta poiché svolgono le funzioni amministrativo-gestionali, il primo, e di broker di servizi, il secondo. Il modello prevede che gli agenti operanti su di una piattaforma vadano ad interfacciarsi con questi due elementi scambiando con loro messaggi, espressi in ACL. Nella visione degli agenti normali, l'AMS ed il DF sono quindi altri agenti: si è già visto in 2.4.1 che esistono ontologie apposite per la comunicazione con l'AMS ed il DF. Per "normale" si intende un agente che non ha funzioni correlate con l'operatività della piattaforma. Non è necessariamente così dal punto di vista implementativo. Comunicare con l'ACL in questo caso non implica l'essere un agente: l'agente funziona sulla piattaforma, con il supporto della piattaforma, ed è composto di codice eseguibile supportato dalla piattaforma. In una piattaforma progettata per codice PERL potranno operare agenti scritti in PERL. In questo caso è possibile che l'implementatore scelga di implementare l'AMS in PERL e di farlo eseguire come processo separato, così che l'AMS stesso abbia le caratteristiche salienti dell'agente, e possa a

ragion veduta essere definito “agente”. Ma è anche possibile che l’AMS venga implementato a livello di piattaforma, a livello dell’interprete PERL, in un “kernel di piattaforma” di tipo monolitico. Un AMS così progettato non può essere definito “agente”, anche perché il linguaggio sufficiente per le comunicazioni con questo elemento è l’SL0, cioè il sottoinsieme minimo dell’SL, e definire agente una entità software che implementa solo tale linguaggio è, a dire il vero, un pò eccessivo. Dipende dai punti di vista. La fig. 13 mostrata in precedenza mette in risalto questi concetti, poiché in essa l’AMS ed il DF sono incorporati nella piattaforma, pertanto stanno ad un livello inferiore agli agenti normali. Ma, costituendo entità distinte, autonome a livello concettuale, e comunicando con agenti normali in ACL, possono essere implementati come agenti.

L’AMS è un elemento obbligatorio e deve esisterne uno solo all’interno di una piattaforma FIPA conforme, anche se questa si estende su più macchine. Le sue funzioni sono quelle di registrare tutti gli agenti attivi (cioè in uno stato diverso da unknown, vedi 2.5.2) presenti nella piattaforma e di rendere pubblico l’elenco degli agenti registrati attraverso un servizio di “pagine bianche”, di dare agli agenti registrati accesso al MTS, in modo che questi possano interagire tra loro e con agenti di altre piattaforme, di gestire il ciclo di vita degli agenti: quindi è l’AMS che si occupa dell’avviamento o della terminazione di un agente. Con servizio di “pagine bianche” (dall’inglese “white pages”) ci si riferisce al fatto che un agente può ottenere informazioni sul come contattare un altro agente, o sul suo proprietario: ci si riferisce quindi ad un servizio simile a quello offertoci dall’elenco telefonico.

Il DF costituisce il cosiddetto servizio di “pagine gialle”: un agente si rivolge al Directory Facilitator per ottenere informazioni su quali siano gli agenti che soddisfano certe condizioni in termini di servizi offerti. Si vedrà più avanti che consente ricerche più avanzate di quelle effettuabili con il directory-service dell’architettura astratta. È comunque compito degli agenti registrarsi presso il DF, affinchè siano visibili nelle ricerche. A tale proposito il documento di specifica precisa come il DF sia obbligato a mantenere dati corretti e aggiornati, ed a effettuare ricerche per gli agenti autorizzati in maniera non discriminatoria. Non è però compito del DF controllare se i dati che possiede sono veritieri o aggiornati, poiché è l’agente registrato che deve chiedere al DF la modifica delle impostazioni di registrazione in caso di cambiamenti che lo riguardino, ne tantomeno è compito del DF controllare quale sia lo stato in cui si trova un agente al momento di una ricerca che lo riguardi. Una piattaforma può prevedere più DF, ed è compito dell’agente richiedere la registrazione a quelli appropriati. È inoltre previsto che un DF si registri come fornitore di tale servizio presso altri DF, così da costituire una federazione.

Del MTS ci si occuperà al punto 2.5.7. Qui si dirà soltanto che, come dovrebbe essere chiaro, è il sistema di trasmissione messaggi tra agenti, sia all’interno che all’esterno della piattaforma, e può collegarsi a più meccanismi di trasporto. La sua funzionalità in più rispetto al message-transport-service consiste nel rendere possibile lo scambio di messaggi anche tra agenti presenti su piattaforme che non hanno sistemi di trasporto in comune. Si vedrà che ciò avviene con un processo che ingloba piattaforme esterne a quelle direttamente interessate alla comunicazione, che hanno funzione di “routing”, per fare un paragone con il linguaggio di rete.

2.5.2 Il ciclo di vita di un agente

In un qualunque istante, un agente esiste ed è in esecuzione nel contesto di una sola piattaforma. Possiamo anche avere un agente funzionante su un sistema operativo distribuito, come Mach o QNX, e che quindi possa procedere nella sua esecuzione su più macchine ([12]), ma solo se anche la piattaforma risulta distribuita su tali macchine. Se sia l'agente sia la piattaforma sulla quale questo risiede supportano la mobilità, è possibile che durante il corso della sua esistenza l'agente in questione migri, per sua spontanea decisione, da una piattaforma ad un'altra. Chiaramente nulla vieta (a parte casi particolari) che su una singola macchina siano in funzione più piattaforme. In ogni caso un agente, durante il corso della sua esistenza e su qualunque piattaforma si trovi in un certo istante, può trovarsi in cinque diversi stati (fig. 14):

- *initiated*: l'agente è stato creato, il suo codice è stato caricato, gli sono state allocate le risorse, la piattaforma che compie queste operazioni sarà la sua HAP: l'agente esiste ma ancora non è cominciata la sua esecuzione;
- *active*: l'agente è in esecuzione
- *suspended*: l'esecuzione è sospesa; l'agente viene sospeso (o decide di sospendersi) per un certo periodo di tempo; verrà risvegliato dalla AP;
- *waiting*: l'esecuzione è sospesa; l'agente attende che si verifichi un certo evento per continuare la sua esecuzione; sarà l'AP a risvegliarlo appena si verifica l'evento;
- *transit*: solo gli agenti mobili possono entrare in questo stato; l'esecuzione è sospesa; si tratterà di questo stato al punto 2.5.3;

Inoltre è previsto lo stato che si riferisce alla non esistenza di un agente: questi si trova nello stato *unknown* prima di essere creato e dopo aver finito la sua esecuzione o essere stato distrutto.
La fig. 14 mostra gli stati ora descritti e le possibili transizioni tra questi.
Segue una descrizione delle transizioni:

- *create*: l'agente viene istanziato, passando da uno stato di non esistenza (unknown) allo stato initiated; a decidere la creazione del nuovo agente può essere un utente o un altro agente;
- *invoke*: l'agente in stato initiated viene invocato: coincia la sua esecuzione, passando così allo

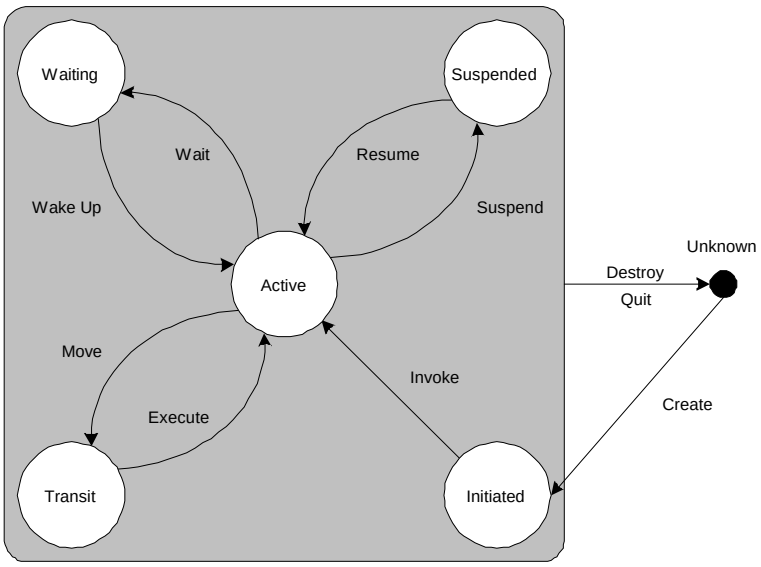


Fig 14 [14]: automa a stati finiti del ciclo di vita di un agente

stato active; la decisione dipende sempre da un utente o da un altro agente;

- *quit*: l'agente vuole terminare, oppure per qualche motivo è la piattaforma ospite che chiede all'agente in esecuzione di terminare (soft break). In quest'ultimo caso l'agente è libero di ignorare la richiesta e rimanere in esecuzione;
- *destroy*: la piattaforma ospite decide di interrompere forzatamente l'esecuzione di un agente, in poche parole di distruggerlo (hard break); l'agente non può ignorarlo; l'operazione è avviata dall'AMS (magari in seguito ad una richiesta utente);
- *suspend*: l'agente viene bloccato, per volontà sua o dell'AMS;
- *resume*: l'aggente viene riavviato: riprende l'esecuzione al punto in cui era stato sospeso; ovviamente solo l'AMS può decidere un resume;
- *wait*: l'agente decide in proprio di mettersi in pausa mentre aspetta un evento;
- *wake up*: l'agente viene risvegliato dallo stato di attesa; ovviamente solo l'AMS può decidere un wake up;
- *move*: l'agente è messo in stato di transizione per il suo trasferiento su di un'altra piattaforma; è proprio l'agente a richiedere l'azione, che è possibile solo su piattaforme con supporto per la mobilità; maggiori dettagli al punto 2.5.3;
- *execute*: l'agente viene risvegliato dallo stato transitorio richiesto dallo spostamento interpiattaforma; ad effettuare l'operazione è l'AMS della piattaforma di destinazione, se l'operazione di migrazione è andata a buon fine; anche in questo caso maggiori chiarimenti saranno dati al punto 2.5.3;

Affinchè siano possibili tali transizioni, l'AMS deve poter istruire il kernel[4] della piattaforma sulle seguenti operazioni: *create agent*, *invoke agent*, *suspend agent*, *resume agent execution*, *terminate agent*, *execute agent*, *resource management*. Tutti gli agenti devono inoltre supportare la funzione *quit*, scambiata con l'AMS attraverso l'ACL, con la quale l'agente, terminando la sua esecuzione, chiede all'AMS di liberare le risorse ad esso associate, o viceversa l'AMS chiede all'agente di terminare al più presto possibile la sua esecuzione.

FIPA obbliga il progettista ad implementare primitive bufferizzate ([12]) per l'ingresso dei messaggi agli agenti: quando un agente si trova negli stati initiated, waiting e suspended, il MTS deve accodare i messaggi a lui destinati, così che al momento del risveglio dell'agente, questi possa riceverli; ma è anche possibile che, negli stessi casi, sia definito un indirizzo di forward, cioè un indirizzo cui ulteriormente spedire i messaggi arrivati mentre l'agente non è in esecuzione. Nel caso in cui l'agente di destinazione si trovi nello stato di transito è invece possibile che: a) se il messaggio è diretto alla vecchia piattaforma, questa possa o accodarlo, nel caso in cui l'operazione di migrazione non vada a buon fine, o fare il forward (rispedirlo) al nuovo indirizzo; b) se il messaggio è diretto alla nuova piattaforma, questa deve accodarlo fintantochè l'agente non risulti in stato di esecuzione. Ma anche nel caso in cui un agente sia inesistente (stato unknown), è possibile che il MTS, invece di riggettarlo, decida di bufferizzare il messaggio nell'aspettativa che venga creato un agente con caratteristiche coincidenti a quelle del destinatario, ma questo dipende anche dalle caratteristiche del messaggio.

[4] Con il termine kernel ci si riferisce al nucleo di gestione dei processi in una AP. Non è compito di FIPA dare direttive sui modi di realizzazione di tale componente in una piattaforma reale. Tale realizzazione dipende infatti strettamente dalla base hardware/software sulla quale operano la piattaforma stessa e, su di essa, gli agenti. Non è, d'altra parte, neanche compito

2.5.3 Il supporto della mobilità

Nel prendere decisioni riguardo le specifiche relative alla mobilità, i membri FIPA hanno constatato che sulle piattaforme per agenti mobili esistenti, questa viene gestita in modi diversi. Per garantire il massimo della flessibilità, si è deciso di rendere possibili due protocolli diversi per la mobilità, con diversi punti di vista della mobilità stessa sia per l'agente che per le piattaforme di partenza e destinazione, facendo in modo che questi possano coesistere. Il modello per la mobilità consta quindi di:

- due protocolli possibili: *simple mobility* (mobilità semplice) e *full mobility* (mobilità piena);
- tre operazioni possibili: *agent migration* (migrazione), *agent cloning* (clonazione), ed *agent invocation* (invocazione);
- uno stato dedicato di esecuzione dell'agente (vedi 2.5.2): transition;
- due transizioni di stato di esecuzione (vedi 2.5.2): move, iniziata dall'agente, ed execute, iniziata dall'AMS della piattafora di destinazione;

Per protocollo si intende lo scambio di messaggi necessario tra piattaforme, affinchè sia possibile realizzare la mobilità. Nella visione FIPA, questa consiste nel poter fare migrare un agente da una piattafora ad un'altra compatibile, nel clonare un agente in modo che una copia esatta di questo venga attivata su una piattaforma di destinazione, o nel poter invocare un agente su un'altra piattaforma. Se termini come migrazione e clonazione sono ovvi, non è così per il termine invocazione: ci si riferisce al fatto che il codice di un agente (magari anche non mobile) viene copiato ed istanziato, diventa perciò un agente in esecuzione, su un'altra piattaforma rispetto a quella dove è memorizzato fisicamente. Il che può risultare di grande utilità per esempio per sistemi che prevedono agenti controllori principali che gestiscono una schiera di agenti satelliti, avviati su piattaforme remote.

Il protocollo di mobilità semplice, schematizzato in fig. 15, prevede che sia la piattaforma dev'è situato l'agente prima di muoversi a gestire i dettagli del processo di movimento. L'agente semplicemente chiede all'AP, attraverso l'operazione move, di creare in un istante preciso[5] una

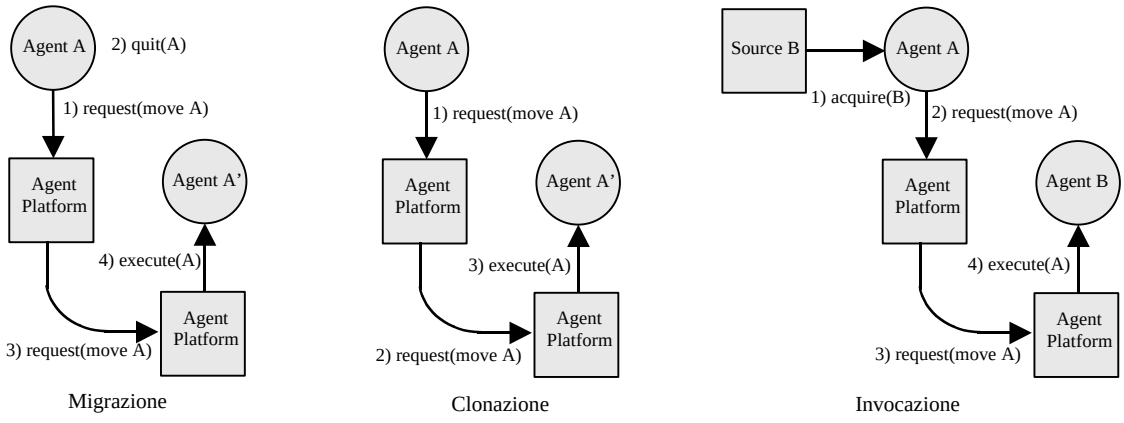


Fig. 15 [15]: realizzazione delle operazioni di mobilità con il protocollo di mobilità semplice.

---

del singolo agente preoccuparsi di eventuali suoi rapporti con il kernel, dal momento che le funzioni di gestione sono trattate con l'AMS attraverso normali messaggi ACL.

[5] L'agente deve prima essere fermato, per potere in seguito "congelare" il suo stato interno. Si può altrimenti dire che bisogna effettuare "un'istantanea" dello stato interno, che deve quindi essere acquisito dal kernel in maniera completa in un singolo istante. Sinteticamente: lo stato interno acquisito deve essere consistente.

copia esatta di se stesso (un clone) sia in termini di codice che di stato interno (strutture dati). É il clone che si troverà nello stato transit subito dopo la sua creazione, mentre l'agente originario continuerà a rimanere active. Il clone viene spostato dall'AP di partenza a quella di destinazione, e qui viene portato in stato active dall'AMS, che richiede al suo kernel una transizione execute. Con questo protocollo, l'agente realizza l'agent migration se termina sulla piattaforma di partenza subito dopo chiamata la move. Viceversa, se continua ad esistere su entrambe le piattaforme, l'agente realizza l'agent cloning. L'agent invocation si realizza se l'agente non chiede di creare un clone di se stesso, bensì acquisisce il codice di un agente (unknown) e chiede una move di questo.

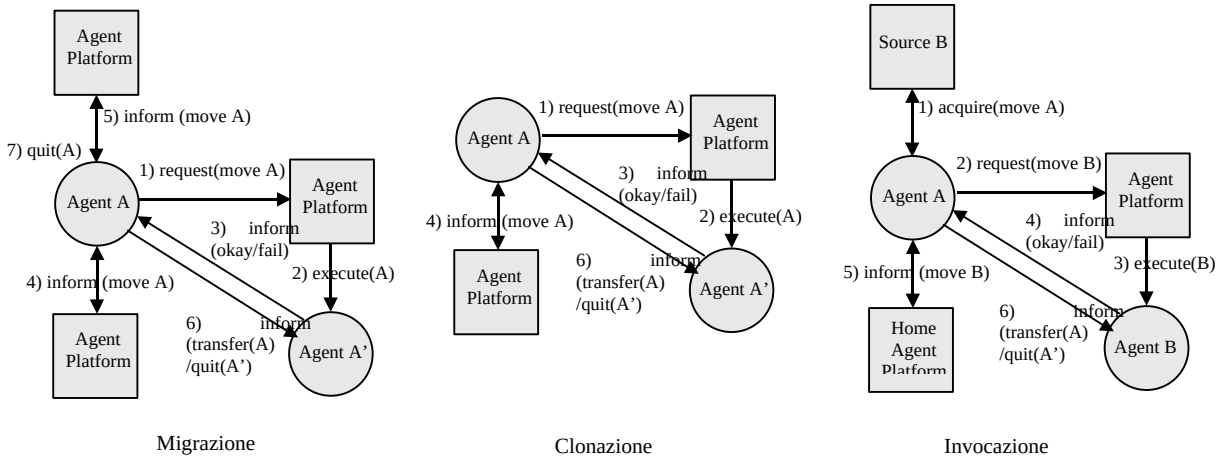


Fig. 16 [15]: realizzazione delle operazioni di mobilità con il protocollo di mobilità semplice.

Il protocollo di mobilità piena, schematizzato in fig. 16, permette invece all'agente di controllare e di gestire in proprio le operazioni di mobilità. In questo caso è l'agente che comunica direttamente con l'AP remota, ed in particolare con l'AMS, richiedendo una move. Viene creato un clone dell'agente, il quale si troverà nello stato di transit, mentre l'agente originario continua ad essere active. Il clone viene trasferito sulla piattaforma di destinazione, e qui l'AMS richiede al kernel un'execute, per portare il clone in esecuzione. Il codice dovrà essere progettato in modo che il clone, non appena risvegliato dallo stato di transit, informi l'agente originario che tutto si è svolto correttamente. L'agente originario potrà in seguito informare la sua ed altre piattaforme dell'evento (per esempio potrà avvertire i DF, così come potra farlo il clone) ed in seguito deciderà se terminare, realizzando l'agent migration, o rimanere in esecuzione, realizzando l'agent cloning: in entrambi i casi informerà il clone della decisione presa. L'agent invocation è realizzato richiedendo una move del codice di un altro agente (in stato unknown) e procedendo come visto in precedenza.

È chiaro che il secondo protocollo garantisce maggiore controllo da parte dell'agente sulle operazioni di mobilità, ed inoltre semplifa la piattaforma, che si trova a dover fornire soltanto servizi di basso livello, ma ha il notevole svantaggio di aumentare la complessità interna di un agente che ne fa uso: il clone deve informare del successo dell'operazione l'agente originario, il che implica che bisogna realizzare il codice dell'agente in modo tale che il clone riconosca di essere tale e non l'agente originario, e cioè che l'esecuzione dei due agenti prenda strade diverse dal momento cui viene fatta la richiesta di move (in modo similare un processo Unix può duplicarsi, con la chiamata di sistema fork, ed il doppione prende un percorso di esecuzione diverso da quello preso dal

processo originale). Ovviamente l'agente che vuole usufruire dei servizi di mobilità, deve comunicare quale protocollo intende usare.
Dal momento che ambedue i protocolli possono essere utili in diversi contesti, e per garantire un pieno supporto a tutti gli agenti mobili, le piattaforme FIPA devono implementarli entrambi.
Va notato il fatti che i progettisti FIPA sono riusciti ad ottenere uguali funzionalità (operazioni) attraverso protocolli diversi (che danno punti di vista diversi agli elementi in gioco), basati su azioni uguali (move ed execute). Dal punto di vista dei successivi livelli di astrazione, questo vuol dire che il cambiamento del livello intermedio, quello di protocollo, non necessità ne di un cambiamento del livello inferiore, quello delle azioni rese disponibili dal sistema, ne di un cambiamento del livello superiore, quello delle funzionalità possibili per un agente mobile. Niente di meglio per favorire la flessibilità.

2.5.4 L'agent management: AMS e DF

Si entrerà ora nel merito dello scambio di messaggi tra agenti e AMS/DF e dell'ontologia FIPA a questo necessaria: FIPA-Agent-Management. In 2.4.1 si è chiarito il modo in cui vengono definite le ontologie FIPA ed il lunguaggio usato per esprimere i contenuti nel loro dominio: l'SL0.
Sicuramente è opportuno precisare in che modo il modello si occupa dell'identificatione e della localizzazione di un agente. Allo scopo viene dedicato un frame che si occupa di ambedue le cose: *agent-identifier*, normalmente chiamato AID. Esso contiene tre parametri: *name* è il GUID dell'agente; *address* è una lista ordinata di indirizzi espressi sotto forma di URL; *resolvers* è una lista ordinata di risolutori, ognuno a sua volta indicato per mezzo di un AID. Verrà chiarito cos'è e qual è lo scopo di un risolutore al punto 2.5.8. Poiché il nome identifica già un agente in modo univoco, questo è il solo parametro mandatory dell'AID. È importante considerare che le liste sono in questo caso ordinate e che, a tal proposito, l'SL0 distingue con le due parole chiave *set* e *sequence* i concetti di insieme semplice e lista ordinata rispettivamente. Il nome di un agente è solitamente del tipo *nomespecificoagente@hap*, dove hap è ovviamente riferito all'agente in questione. Un nome lecito è quindi agent-a@foo.com. L'AMS ed il DF hanno il nome fisso ams@piattaforma e df@piattaforma rispettivamente, che costituiscono valori riservati del parametro name dall'AID.
Diamo un esempio di possibile AID:

```
(agent-identifier
  :name agent-a@foo.com
  :address (sequence
    http://foo.com/acc
    iiop://foo.com/corba/acc
    http://foo.com/wap/agent_a)
  :resolvers (sequence
    (agent-identifier
      :name ams@fip.com
      :address (sequence iiop://fip.com/acc))))
```

Il concetto di ACC verrà chiarito successivamente, ma si può rapidamente dire che esso costituisce per agenti interni alla piattaforma l'accesso standard reso disponibile dalla piattaforma stessa da e verso l'esterno. Un agente può anche accedere all'esterno della piattaforma con mezzi propri. È quanto fa agent-a@foo.com, che oltre ad essere raggiungibile attraverso i mezzi standard della piattaforma (guardare le righe contenenti acc), è anche raggiungibile attraverso il protocollo WAP.

L'agente dovrà inviare informazioni all'AMS e al DF se vuole operare realmente. L'AMS costituisce il servizio di pagine bianche e per ogni agente conserverà il relativo AID, il suo proprietario, e suo stato di esecuzione. Il *ams-agent-description* è il frame che contiene le informazioni che un agente invia all'AMS, e dispone di un parametro *name*, contenente l'AID, e dei parametri *ownership* e *state*, che contengono sotto forma di stringhe rispettivamente il proprietario e lo stato. Gli stati in cui può trovarsi un agente sono quelli elencati in 2.5.2 (che costituiscono valori riservati del parametro *state*). Il DF costituisce il servizio di pagine gialle, ed in quanto tale dovrà rendere pubblici i servizi messi a disposizione dall'agente. Il frame *service-description* serve proprio per definire un certo servizio, e dispone dei parametri:

- *name*: una stringa, contenente il nome dato al servizio;
- *type*: una stringa, contenente la tipologia del servizio considerato; un esempio è *fipa-df*;
- *protocol*: un insieme di stringhe, contenenti i nomi dei protocolli di interazione necessari per portare a termine il servizio;
- *ontology*: un insieme di stringhe, contenenti i nomi delle ontologie cui far riferimento durante lo scambio messaggi riguardante il servizio;
- *language*: un insieme di stringhe, contenenti i nomi dei content language usati nel corso dello scambio messaggi riguardante il servizio;
- *ownership*: una stringa, indicante il proprietario del servizio;
- *properties*: un insieme di *property*, con i quali si possono indicare proprietà opzionali dipendenti dal tipo di servizio.

Tali parametri sono tutti opzionali.

Property è un frame che nell'ontologia FIPA-agent-management rappresenta il concetto di coppia chiave-valore, e pertanto ha due parametri: name e value, ripettivamente la chiave ed il valore.

Detto ciò, un agente che voglia registrarsi, o modificare la su registrazione, presso il DF, invierà a quest'ultimo le informazioni contenute nel frame *df-agent-description*, i cui parametri sono:

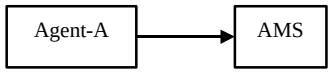
- *name*: l'AID dell'agente;
- *services*: un insieme di *service-description*;
- *protocol*: un insieme di stringe, contenenti i nomi di tutti protocolli di interazione supportati dall'agente (diversamente dall'equivalente campo del *service-description*, che si riferisce ai protocolli necessari per il singolo servizio);
- *ontology*: un insieme di stringhe, contenenti i nomi delle ontologie cui far riferimento durante le comunicazioni con l'agente (diversamente dall'equivalente campo del *service-description*, che si riferisce alle ontologie necessarie per il singolo servizio);
- *language*: un insieme di stringhe, contenenti i nomi dei content language supportati dall'agente.

Le azioni che un agente può richiedere all'AMS ed al DF derivano direttamente dall'architettura astratta (2.3.2 e 2.3.3), e dunque sono: *register*, con cui un agente chiede di ragistrarsi presso l'AMS o il DF, per cui il dominio sarà ams-agent-description nel primo caso o un df-agent-description nel secondo, e non è previsto alcun risultato (range); *deregister*, con cui un'agente chiede la cancellazione della sua registrazione presso l'AMS o il DF, per cui dominio e range saranno uguali all'operazione di registraione; *modify*, con cui un agente chiede la modifica delle informazioni di registrazione all'AMS o al DF, per cui dominio e range saranno uguali all'operazione di registrazione; *search*, con cui un agente pone un'interrogazione all'AMS o al DF: in questo caso il

dominio è una coppia composta da un'*object-description-template* e da un *search-constraints*, ed il range è un insieme di ams-agent-description o di df-agent-description, a seconda che l'interrogazione sia posta all'AMS o al DF rispettivamente. Un'object-description-template altro non è che un ams-agent-description o un df-agent-description, sempre a seconda che l'interrogazione sia fatta all'AMS o al DF, contenente soltanto i parametri conosciuti al momento dell'interrogazione: gli ams/df-agent-description ritornati dall'operazione search, saranno quelli che soddisfano i requisiti richiesti dall'object-description-template.

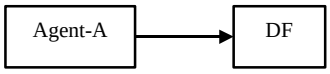
Un object-description-template è quindi un particolare tipo di ams-agent-description o di df-agent-description. Questi due frame possono contenere vari frame in modo innestato, come visibile in fig. 17. Inoltre sono presenti sequenze ed insiemi. Ecco perché in 2.3.2 si è detto che i meccanismi di ricerca in un'architettura concreta devono essere più complessi di quanto sia il directory-service dell'architettura astratta: le ricerche vengono spesso svolte su livelli multipli di innesto, e il template è da considerarsi un sottoinsieme minimo che deve essere presente, con tutti i frame, parametri e relativi valori, in un ams/df-agent-description affinchè questi venga considerato corrispondente al template stesso. Affinchè un set di un ams/df-agent-description corrisponda a quello relativo nel template di ricerca, è necessario che ogni elemento del set di quest'ultimo sia anche presente nel set del primo. Affinchè, infine, una sequence di un ams/df-agent-description corrisponda a quella relativa nel template di ricerca, è necessario che gli elementi nel primo corrispondano a quelli del secondo, nello stesso ordine. Ovviamente l'insieme di ritorno di un'interrogazione può anche essere vuoto, se non esistono registrazioni che soddisfano i requisiti richiesti. Il search-constraints è un ulteriore frame dell'ontologia FIPA-agent-management, che da informazioni all'AMS/DF su come svolgere la ricerca. Contiene due parametri entrambi opzionali: *max-depth*, di tipo *integer*, indicante il massimo livello di propagazione della ricerca nella federazione di AMS o DF[6]; *max-result*, di tipo

a)




```
(request  // CA
 . . .
 :content
  (action
   (agent-identifier   //AID di chi deve svolgere l'operazione
    :name ams@foo.com
    :addresses (sequence iiop://foo.com/acc))
   (register
    (ams-agent-description  //dominio di register per l'AMS
     :name
      (agent-identifier
        :name agent-a@foo.com
        :addresses (sequence iiop://foo.com/acc))
     :state active
     :ownership foo_inc.))) //chiude fino a content
. . .)
```

b)




```
(request      //agent-a richiede …
 . . .
 :content
  (action
   (qui l'AID di chi svolge l'azione, qui df@foo.com)  //… al DF …
   (register          //… di svolgere un'operazione di registrazione
    (df-agent-description
     :name
      (qui l'AID dell'agente che si registra)
     :protocol (set FIPA-Request Application-Protocol)
     :ontology (set FIPA-agent-management meeting-scheduler)
     :language (set FIPA-SL0 KIF)
     :services (set       //l'agente svolge un solo servizio
      (service-description
       :name profiling
       :type user-profiling
       :ontology (set meeting-scheduler)
       :properties (set   //si indicano due proprietà del servizio
        (property
         :name learning-algorithm
         :value BBN)
        (property
         :name max-nodes
         :value 1000000))))  ///chiude fino a set di services
    )))  //chiude fino a content
. . .)
```

Fig. 17: Esempi di registrazione presso (a) l'AMS ed il (b) DF. Il linguaggio usato è l'SL0.Vengono visualizzati solo il CA ed il content del messaggio. A destra in diverso carattere sono riportati commenti in notazione C++. Si noti che l'innesto dei frame ed i parametri di questi seguono le descrizioni dell'ontologia. Se la richiesta è corretta, seguirà da parte dell'AMS/DF un messaggio con CA agree. In seguito, se la registrazione viene effettuata con successo, seguirà un ulteriore messaggio con CA inform per informare il richiedente del buon esito dell'operazione, come indicato dal protocollo FIPA-request

[6] Un AMS può registrarsi presso altri AMS, e così anche per i DF. Un AMS/DF oltre a fare la ricerca sui propri dati, chiederà di svolgere la ricerca anche agli AMS/DF registrati presso di sé, e questi faranno la stessa cosa, per un numero di livelli definito in max-depth.

integer, indicante il numero massimo di elementi richiesti nell'insieme di ritorno della search.
Le azioni di cui ora si è detto vengono normalmente gestite dal protocollo FIPA-request.
È possibile chiedere all'AMS quali siano le funzionalità della piattaforma. A tal proposito l'AMS supporta l'operazione *get-description*, che non ha argomenti di dominio, ma come range ha un *ap-description*. Quest'ultimo è un ulteriore frame che permette di descrivere le proprietà di un AP. Dispone dei seguenti parametri:

- *name*: una stringa, contenente il noe della piattaforma; è l'unico parametro obbligatorio di ap-description;
- *dynamic*: un booleano,;
- *mobility*.un booleano, vero se l'AP supporta la mobilità;
- *transport-profile*: un *ap-transport-description*, frame che, come si vedrà in 2.5.8, contiene una descrizione dei mezzi di trasporto supportati dall'MTS.

Tutti gli agenti devono supportare l'operazione *quit* di cui si è già parlato in precedenza. Viene usata sia dall'AMS per chiedere la terminazione dell'agente che la riceve, sia da un qualsiasi agente per chiedere all'AMS la terminazione di un altro agente. L'operazione *quit* non ha range, e il suo dominio è l'agent-identifier identificante l'agente da terminare.
Se si verificano errori nel corso dello svolgimento delle azioni fin qui viste, vengono generate delle eccezioni, argomento del punto 2.5.6.

2.5.5 L'agent management: mobilità

L'ontologia FIPA-agent-management copre anche l'argomento della mobilità. Un agente mobile potrà migrare da una piattaforma ad un'altra solo se ambedue supportano la mobilità, e se sono compatibili con l'agente considerato. I concetti essenziali che stanno dietro quest'ultimo termine sono tre: il sistema, il codice eseguibile, il sistema operativo. Con il primo termine, "sistema", ci si riferisce al sistema attraverso cui è implementa la mobilità del codice, che per esempio potrebbe sfruttare la serializzazione di oggetti Java, oppure, in maniera sicuramente più rozza, ricorrere ad un dump su file e successivo trasferimento. Esempi di sistemi di mobilità reali sono l'Aglets, il Mole e l'AgentTcl. Il secondo termine ha un significato ovvio: un agente può muoversi tra piattaforme che gestiscono il suo tipo di codice eseguibile: un agente Java sarà eseguito solo su piattaforme progettate per eseguire byte-code Java; un agente in formato ELF i386, potrà essere eseguito solo su piattaforme che supportano codice eseguibile per 80386 (quindi basate su questo microprocessore); ecc. Il terzo termine ha anch'esso significato ovvio: il sistema operativo può essere un fattore condizionante la possibilità di esecuzione di un agente (così come di un processo). Questo vale più nel caso di agenti in forma di codice macchina, programmati su uno specifico sistema operativo, che nel caso di agenti programmati in linguaggi interpretati -quali Perl o Tcl- o in codice intermedio tipo byte-code -quali Java o April-, progettati per essere nella maggior parte dei casi indipendenti dall'hardware e dal sistema operativo. In generale comunque le tre caratteristiche devono essere comuni alle piattaforme e concordi con quelle dell'agente.
L'insieme delle caratteristiche sistema di mobilità–codice eseguibile–sistema operativo (da quanto detto quest'ultimo può in alcuni casi non essere specificato) è detto *profilo* dell'agente. Il frame che ne esprime il concetto è *mobile-agent-profile*. I suoi parametri sono:

- *system*, obbligatorio, di tipo *mobile-agent-sytem*, esprime il sistema di mobilità;

- *language*, obbligatorio, di tipo *mobile-agent-language*, esprime il tipo di codice eseguibil dell'agente;
- *os*, opzionale, di tipo *mobile-agent-os*, esprime il sistema operativo su cui l'agente deve funzionare.

Ognuno dei tre parametri è espresso con un frame appropriato.

Il frame mobile-agent-system dispone di quattro parametri, dei quali i primi due obbligatori: *name*, una stringa, contiene il nome del sistema di mobilità; *major-version* e *minor-version*, stringhe, contengono rispettivamente il massimo ed il minimo numero di versione del sistema supportata dall'agente; *dependencies*, insieme di *property*, esprime ulteriori dipendenze richieste al sistema di mobilità.

Il frame mobile-agent-language dispone di sei parametri: *name*, una stringa, esprime il nome convenzionale del tipo di codice eseguibile; *major-version* e *minor-version*, stringhe, contengono rispettivamente il massimo ed il minimo numero di versione supportata del codice eseguibile; *format*, una stringa, esprime il formato del codice eseguibile (può essere binario nel caso di codice macchina o byte-code, testuale nel caso di codice da interpretare, ed altro); *filter*; *dependencies*, insieme di *property*, esprime ulteriori dipendenze richieste al tipo di codice eseguibile. Il nome, il numero massimo di versione e il formato sono obbligatori.

Il frame mobile-agent-os dispone di cinque parametri, dei quali i primi due obbligatori: *name*, una stringa, contiene il nome del sistea operativo; *major-version* e *minor-version*, stringhe, contengono rispettivamente il massimo ed il minimo numero di versione supportata del sistema operativo; *hardware*, una stringa, esprime l'hardware del sistema operativo; *dependencies*, insieme di *property*, esprime ulteriori dipendenze richieste al sistema operativo.

Come si è visto sono due le azioni concui realizzare la mobilità: move e execute. Di queste solo la move è richiesta dall'agente. L'azione *move* è quindi supportata (e svolta) dall'AMS. Non ha range, ed il suo dominio è un *mobile-agent-description*. Quest'ultimo è un frame che rappresenta l'agente durante lo stato di trasferimento, e dispone di sei parametri:

- *name*: agent-identifier dell'agente da trasferire;
- *profile*: insieme di mobile-agent-profile, che rappresentano i requisiti di mobilità dell'agente;
- *version*: una stringa, contiene la versione dell'agente;
- *protocol*: insieme di stringhe, contenenti i nomi dei protocolli di mobilità supportati daall'agente;
- *code*: binario, contiene il codice eseguibile dell'agente da trasferire;
- *data*: binario, contiene i dati dinamici (lo stato delle strutture dati in esecuzione) dell'agente da trasferire.

L'unico parmetro obbligatorio è l'AID. In particolare, gli ultimi due parametri sono utilizzati se un agente vuole trasferire un altro agente, ed è capace di acquisirne il codice ed i dati prima di chiederne il trasferimento. Ovviamente, nel caso in cui un agente voglia spostare se stesso, sarà l'AP a preoccuparsi dell'acquisizione del codice e dei dati relativi.

Una ulteriore azione supportata dall'AMS che un agente può chiedere è *transfer*. Permette ad un agente di inviare la propria identità insieme con le autorizzazioni relative ad un altro agente. Questo affinchè, ad esempio, un suo clone possa godere di tutti i privilegi associati al su "generatore". Un agente, inoltre, può anche rifiutarsi di accettare l'identità ed autorità che gli vengono inviate

attraverso la *transfer*, ad esempio per motivi di sicurezza. Il dominio della funzione *transfer* è un mobile-agent-description. La funzione non ha range.
Se si verificano errori nel corso dello svolgimento delle azioni fin qui viste, vengono generate delle eccezioni, argomento del prossimo paragrafo.

2.5.6 Eccezioni

Si è detto che le ontologie FIPA definiscono frame e functions: concetti ed azioni. Le eccezioni costituiscono un'ulteriore categoria di oggetti definiti dalle ontologie FIPA, e sono necessari in quanto gestiscono le condizioni di errore che possono verificarsi nel corso dello svolgiento di una azione. Nella discussione sulle operazioni rese disponibili dai due servizi previsti dall'architettura astratta, il directory-service ed il message-transport-service, si è infatti detto che ognuna di queste operazioni può incappare in errore. Elemento questo che dipende dall'operazione considerata e dall'anomalia che si verifica (vedi 2.3.2 e 2.3.3). Il protocollo utilizzato deve permettere la generazione di un messaggio, diretto verso il richiedente dell'azione, con CA adatto alla descrizione dell'errore verificatosi durante lo svolgimento dell'azione stessa. Si è fatto in precedenza l'esempio di una richiesta di registrazione presso il DF da parte di un agente, azione questa controllata normalmente dal protocollo FIPA-request. In fase di registrazione possono aversi questi errori (può essere utile confrontarli con quelli proposti in 2.3.2): la richiesta di registrazione non è corretta, ed in questo caso il protocollo prevede l'invio di un messaggio con CA not-understood o refuse a seconda di quale tipo di errore sia presente nella richiesta; il richiedente non ha il permesso per richiedere l'azione e questa viene rifiutata, ed in questo caso il protocollo prevede l'invio di un messaggio con CA refuse; il richiedente è già registrato e l'azione fallisce, in questo caso il protocollo prevede l'invio di un messaggio con CA failure. In particolare quest'ultimo viene inviato dopo (vedere schema in fig. 12) l'agree alla richiesta di registrazione, poiché l'errore avviene durante l'azione e non in fase preliminare: nell'ultimo caso la richiesta risulta valida, e perciò viene inviato l'agree; soltanto in seguito si scopre che l'agente è già registrato, e viene inviato il failure.
In generale gli errori possibili nel corso di una richiesta di operazione sono riconducibili a questi tre communicative acts: not-understood, refuse e failure. Questi sono i comunicative acts dei messaggi che tornano indietro al richiedente dell'azione per indicargli l'errore. Per ognuno di questi CA un'ontologia FIPA deve prevedere un insieme di possibili eccezioni, cioè di sottocasi, che possano essere specificati nel content del messaggio d'errore. Ogni eccezione è indicata da un *predicato*, che informa sul tipo di errore, e da un *argomento*, che può servire al richiedente dell'azione per capire cosa ha provocato l'errore
Esamineremo qui le eccezioni dell'ontologia FIPA-agent-management per i tre CA suddetti.
Not-understood è l'errore che si ha quando il CA del messaggio del richiedente non è compreso dal destinatario. FIPA-agent-management dispone di quattro predicati di eccezione per questo CA:

- *unsupported-act* e *unexpected-act*: si hanno quando il ricevente non supporta il CA indicato nella richiesta oppure quando questi è fuori dal contesto o dalla sequenza definita dal protocollo; in ambedue le eccezioni l'argomento è una stringa, che identifica il CA in questione;

- *unsupported-value* e *unrecognised-value*: si hanno quando il valore di un parametro del messaggio[7] non è supportato, o addirittura non è compreso (magari perché l'ontologia indicata nel messaggio non lo prevede proprio) dal ricevente; in ambedue lee eccezioni l'argomento è una stringa, che identifica il nome del parametro il cui valore ha causaato l'eccezione.

Refuse è l'errore che si ha quando l'azione richiesta non è svolta (supportata) dal ricevente, o se il richiedente non ha l'autorizzazione necessaria per chiederla, o ancora se la richiesta è sintatticamente o semanticamente mal specificata. FIPA-agent-management dispone di otto predicati di eccezione per questo CA:

- *unauthorised*: si ha quando il richiedente non è autorizzato a chiedere l'azione; il predicato non ha argomento;
- *unsupported-function*: si ha quando il ricevente non svolge l'azione richiestagli; l'argomento è una stringa identificante la *function* che viene richiesta ma che non è supportata;
- *missing-argument*, *unexpected-argument* e *unexpected-argument-count*: il primo si ha quando manca un argomento obbligatorio al dominio dell'azione richiesta (non confondere l'argomento del dominio dell'azione con l'argomento del predicato dell'eccezione); il secondo quando, al contrario, è presente un argomento non previsto nel dominio dell'azione; il terzo quando, genericamente, il numero di argomenti nel dominio dell'azione non è corretto; i primi due predicati hanno come argomento una stringa identificante l'argomento mancante/aggiunto, il terzo non ha rgomento;
- *missing-parameter*, *unexpected-parameter*, *unrecognised-parameter-value*: (rispettivamente) nel content del messaggio manca un parametro obbligatorio, è presente un parametro non richiesto, il ricevente non comprende il valore di un parametro; l'argomento è, per tutti e tre i predicati, una coppia di stringhe: la prima identifica il frame del parametro errato, la seconda il parametro errato vero e proprio.

Failure è l'errore che si ha quando, dopo aver inviato l'agree al richiedente per informarlo che l'aione verrà eseguita, l'agente incaricato incontra una condizione anomala, e non può terminare l'azione con successo. FIPA-agent-management dispone di sei predicati per questo CA:

- *already-registered*: un agente è già registrato al momento della richiesta di registrazione presso l'AMS/DF; nessun argomento;
- *not-registered*: un agente non è registrato al momento della richiesta di cancellazione presso l'AMS/DF; nessun argomento;
- *internal-error*: si verifica una condizione di errore interno presso l'agente incaricato di svolgere l'azione; l'azione fallisce; l'argomento è una stringa identificante l'errore interno verificatosi;
- *mobility-unsupported*: riguarda la mobilità; si richiedono operazioni di mobilità da/verso una piattaforma che non la supporta; l'azione non può essere portata a termine;
- *profile-unsupported*: riguarda la mobilità; l'AMS ricevente si trova su una piattaforma che non supporta il profilo necessario all'esecuzione dell'agente che si vuole far muovere;
- *agent-already-present*: riguarda la mobilità; l'AMS ricevente ha registrato un agente con lo stesso nome di quello che si vuole far migrare.

[7] Non ci si sta riferendo a parametri presenti all'interno del content, ma ai parametri del messaggio ACL: per essere precisi ci si sta riferendo ai parametri del frame FIPA-ACL-message dell'ontologia FIPA-ACL, discussi in 2.4.2

2.5.7 Il Message Transport System e l'Agent Communication Chanel

Il Message Transport System costituisce l'entità del modello di riferimento che permette lo scambio fisico dei messaggi tra agenti, siano essi funzionanti sulla stessa a su diverse piattaforme. Se l'AMS ed il DF sono punti focali del modello di riferimento, l'MTS ne è la struttura portante poiché, essendo i rapporti tra agenti ed AMS/DF basati su messaggi ACL, è proprio attraverso l'MTS che avviene lo scabio di questi messaggi.

L'MTS è derivato del message-transport-system, dal quale eredita quindi tutte le caratteristiche. Ma a queste aggiunge altre caratteristiche importanti per favorire l'interoperabilità tra piattaforme di tipo diverso. Il modo in cui in sede FIPA si è deciso di progettare questo elemento potrebbe al principio sembrare un po' contorto, ma ci si renderà conto che esistono tutte le premesse e tutti i motivi per arrivare a questa conclusione. Probabilmente la discusione che segue chiarirà concetti importanti selle AP.

Un agente, è già stato detto più volte, è prima di tutto un programma, composto quindi di codice eseguibile, in qualsiasi forma si presenti, cioè per qualsiasi tipo di macchina: fisica o virtuale. In quanto codice eseguibile di tipo generico, un agente può essere in grado, a seconda delle implementazioni, di collegarsi autonomamente con sistemi di trasporto di livello più basso senza che la piattaforma debba saperne niente: non si può proibire ad un programma C (anche questo può costituire un agente) o ad una classe Java di aprire un Socket TCP, e non è detto che la piattaforma debba per forza venirlo a sapere. Ciò implica che un agente può essere in grado di provvedere da solo a connettersi con un altro agente, ed a scambiare messaggi con lui. Una comunicazione svolta in questo modo è quanto di più normale possa esserci per un processo in esecuzione che voglia comunicare con un altro. Questo è un tipo di communicazione possibile tra due agenti. In questa situazione però sono i due agenti che devono provvedere a tutti i dettagli della comunicazione. E, cosa ancora più grave, due agenti potranno comunicare in questo modo solo se hanno un sistema di trasporto in comune. Si è già parlato delle enormi differenze tra i sistemi di trasporto, dell'impossibilità di trovarne uno perfetto per ogni situazione, e dell'importanza dell'interoperabilità tra di questi (vedi 1.1.3).

Il message-transport-service dell'architettura astratta nasce proprio per nascondere i dettagli di comunicazione agli agenti, e per permettere l'utilizzo di più sistemi di comunicazione, per estremizzare l'interoperabilità. Un agente può utilizzare il message-transport-service per inviare

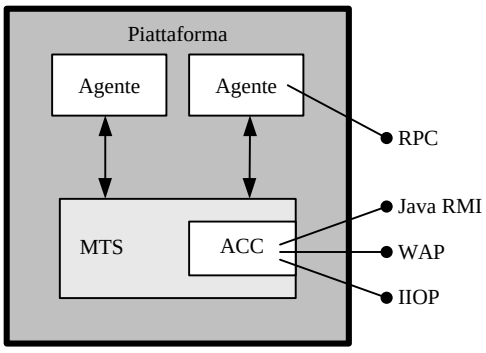


Fig. 18: il meccanismo standard del modello di riferimento per le comunicazioni con l'esterno. È l'ACC ad essere connesso con l'esterno. Gli agenti risultano visibili all'esterno attraverso l'ACC. Nulla vieta comunque ad un agente di collegarsi direttamente ad un sistema di trasporto.

messaggi ad agenti attraverso tutti i sistemi cui il message-transport-system stesso si connette. Ecco che la piattaforma, attraverso il message-transport-service, si connette a più sistemi di trasporto (vedi fig. 18). Il limite di questo sistema è ancora l'interoperabilità: il limite alla comunicazione non è in questo caso imposto dall'agente, che può connettersi solo a pochi dei numerosissimi sistemi di trasporto, ma dalla piattaforma: rimane ancora possibile che due piattaforme non abbiano neppure un sistema di trasporto comune tra loro.

Nasce il Message Transport System. L'elemento principale che distingue quest'ultimo del generico message-transport-service, è l'Agent Communication Chanel, ACC. È un componente che fa parte dell'MTS, ma che il modello di riferimento tratta come l'AS ed il DF: vale a dire che gli agenti comunicano con l'ACC attraverso l'ACL, e pertanto anche l'ACC può esere implementato come agente o può essere "embedded" (innestato) nella piattaforma. Gli agenti che vogliono mandare un messaggio fuori dai confini della piattaforma, lo manderanno all'ACC: da quel momento la consegna del messaggio sarà affidata a lui. L'ACC ha in qualche modo funzioni di router da un sistema di trasporto all'altro, nel senzo che un ACC può prendere dall'esterno un messaggio per cui fa da destinatario intermedio, per poi rispedirlo: due piattaforme non aventi nessun meccanismo di trasporto in comune, potranno scambiarsi messaggi attraverso l'ACC di una terza piattaforma, che abbia i requisiti per dialogare con ognuna delle due. Il processo può, al bisogno, interessare anche più di tre piattaforme: i livelli di routing sono arbitrari.

Non è sbagliato, da quanto appena detto, definire l'ACC come il punto di accesso all'esterno della piattaforma cui appartiene. In 2.5.4 si è visto che spesso negli AID è contenuto l'indirizzo dell'ACC della piattaforma cui appartiene l'agente, piuttosto che l'indirizzo dell'agente stesso: questo perché quasi sempre i messaggi provenienti dall'esterno vengono inviati via ACC all'agente destinatario, e sarà l'ACC a passare il messaggio a quest'ultimo. E ancora: se gli agenti non si collegano in maniera propria (bind) al sistema di trasporto, ma lo fanno attraverso l'MTS, non avranno un loro proprio indirizzo su quel sistema di trasporto, ma saranno raggiunti attraverso l'ACC.

Si elenca ora un insieme di punti importanti per la comprensione del modello:

- il modello non specifica in che modo gli agenti si interfacciano con il proprio[8] ACC: questo dipende dall'architettura concreta di ogni piattaforma; così come, in generale, i metodi per interfacciarsi con l'MTS, e di conseguenza con l'AMS ed il DF;
- il generico sistema di trasporto che consente lo scambio di messaggi tra ACC è detto Message Transport Protocol, o anche MTP; più precisamente: un MTP è un sistema di trasporto cui è collegato l'ACC;
- come si è già detto (2.3.3), il messaggio scambiato tra agenti consiste di un envelope e di un payload, espresso quest'ultimo in una certa codifica. L'architettura astratta indica che nel messaggio ACL (payload) và indicato come destinatario il nome univoco dell'agente di destinazione, e nell'envelope và indicato come destinatario l'indirizzo dell'agente di destinazione. Il modello di riferimento invece prevede, senza entrare in conflitto con l'architettura astratta, che sia nel payload che nell'envelope, l'indicazione del destinatario sia data mediante un AID, che contiene il nome, gli indirizzi ed i risolutori dell'agente di destinazione;

[8] Quello della piattaforma su cui si trovano

- il payload rimane assolutamente immutato, passando da un ACC all'altro; all'envelope, che è costituito da un insieme di coppie parametro/valore associato, vengono invece aggiunte di volta in volta alcune voci, ma senza mai eliminare nulla;
- per svolgere la sua funzione, l'ACC deve basarsi solo sull'envelope, ed inoltre può aver bisogno di ricorrere a servizi dell'AMS o del DF; inoltre l'ACC può incorporare capacità di buffering dei messaggi per permettere funzioni come quelle viste in 2.5.2;
- un messaggio può essere di tipo multicast, cioè con più destinazioni.

2.5.8 ACC ed envelope

I campi previsti in *envelope* sono i seguenti:

- *to*: sequenza di agent-identifier; è scritto del mittente, e contiene l'insieme degli AID di tutte le destinazioni (il messaggio può essere multicast);
- *from*: agent-identifier; è scritto dal mittente, e contiene l'AID di questo; può essere usato dagli ACC interessati alla comunicazione per ritornare messaggi di errore in caso di mancata consegna del messaggio;
- *comments*: una stringa; è scritto dal mittente ed assolutamente opzionale;
- *acl-representation*: un stringa; è scritto dal mittente; come si è già detto il mittente deve comunicare qual è il meccanismo di rappresentazione dell'ACL al destinatario;
- *payload-length*: una stringa; è scritto dal mittente o dagli ACC e contiene la dimensione del payload; conviene usarlo perché l'envelope può essere modificato passando per uno o più ACC; conoscere la dimensione del payload (costante) vuol dire conoscere le dimensioni dell'envelope (dal momento che la dimensione totale del messaggio è resa nota dal sottostante sistema di trasporto), e questo può aiutare gli ACC nel parsing dell'envelope stesso;
- *payload-encoding*: una stringa; è scritto dal mittente, e serve a comunicare se il payload è stato codificato in qualche maniera (compressione, crittografia, …) prima di essere impacchettato con l'envelope e spedito; valori riservati sono per esempio *US-ASCII*, *ISO-8859-1…9*, *UTF-8*, *EUC-JP* o *ISO-2022-JP*;
- *date*: data/ora, che nelle piattaforme è rappresentata in standard ISO; è scritto dal mittente e contiene, dal suo punto di vista, il timestamp di invio del messaggio;
- *encrypted*: sequenza di stringhe; è scritto dal mittente e contiene informazioni utili per decrittografare il messaggio;
- *intended-receiver*: sequenza di agent-identifier; è scritto da uno o più ACC attraverso cui passa il messaggio, per cui può essere presente più volte nell'envelope, e contiene un'insieme di uno o più AID; costituisce uno dei punti fondamentali del modello dell'MTS; per evitare confusione, maggiori dettagli saranno detti tra breve;
- *received*: *received-object* (se ne parla più avanti); scritto da tutti gli ACC attraverso cui passa il messaggio; ognuno di essi, infatti, aggiunge un ulteriore parametro di questo tipo: l'insieme dei parametri received informa sul percorso preso dal messaggio per giungere da mittente al destinatario;
- *transport-behaviour*: tipo non definito; è scritto dal mittente, che così comunica all'MTS i requisiti di trasporto richiesti per l'invio del messaggio: può per esempio richiedere connessioni

low-delay, oppure high-bandwidth; gli ACC cercheranno di soddisfare tali requisiti, ma in caso di impossibilità genereranno un errore.

Si precisa che *envelope*I è anche un frame definito in FIPA-agent-management[9], con tutti i parametri su citati. I parametri obbligatori nell'envelope sono to, from, date e acl-representation. Possono anche essere presenti dei parametri dipendenti dall'implementazione, che se non compresi dal destinatario saranno semplicemente ignorati. I parametri intended-receiver e received possono, se necessario, essere presenti più volte. Ogni evenienza di uno di questi due parametri deve avere un valore diverso dal valore di ogni altra. La cosa importante è che l'inserimento di ogni evenienza deve essere fatto in modo che sia deducibile l'ordine temporale in cui gli inserimenti stessi vengono effettuati.

È d'obbligo a questo punto chiarire il concetto di risolutore. Può capitare che il mittente non conosca l'indirizzo dell'agente, o dell'ACC della sua piattaforma, che vuole raggiungere. Un AID può così contenere, invece dell'indirizzo vero e proprio del destinatario, l'AID di un servizio di risoluzione dei nomi: un agente cioè che sappia collegare il nome con un indirizzo corretto cui spedire i messaggi. Il mittente, così facendo e se utilizza l'ACC per la spedizione, non si preoccupa neanche di conoscere l'indirizzo del destinatario, ma comunica all'ACC di chiederlo ad un risolutore dei nomi. Sarà l'ACC a preoccuparsi di tale richiesta e, una volta venuto a conoscenza dell'indirizzo del destinatario, di inviare il messaggio. Il modo in cui la richiesta di risoluzione viene effettuata non è comunque standardizzato da FIPA.

Il processo di invio del messaggio dipende da come è identificato il destinatario nell'AID del campo to. Il messaggio viene comunque inviato dal mittente sempre senza alcun campo intended-receiver.

Ricevendo un messaggio, un ACC acquisisce l'AID del destinatario: se non esiste alcun campo *intended-receiver*, il campo preso in considerazione è *to*, altrimenti sarà l'ultimo *intended-receiver* scritto nell'envelope ad essere considerato; dopodicchè aggiunge nell'envelope un parametro *received* indicante il passaggio del messaggio anche attraverso di lui. Acquisito l'AID del destinatario, l'ACC considerato è in grado di stabilire se questi opera sulla sua stessa piattaforma o su un'altra:

- nel primo caso (fig.19) viene effettuata la consegna del messaggio al destinatario con i metodi standard messi a disposizione dalla piattaforma, previa aggiunta nell'envelope di un parametro *received*, ed indicante il passaggio del messaggio anche attraverso di lui;

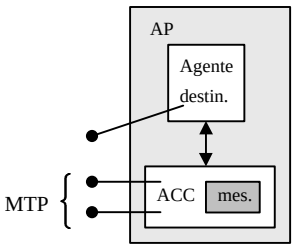


Fig. 19: l'ACC consegna il messaggio all'interno della piattaforma.

- nel secondo caso (fig. 20) l'ACC dovrà inviare il messaggio usando un qualche sistema di trasporto tra quelli indicati nell'address dell'AID, ed in questo caso è anche possibile (oltrechè

[9] Allo stesso modo il concetto di messaggio è definito nell'ontologia FIPA-ACL dal frame FIPA-ACL-Message.

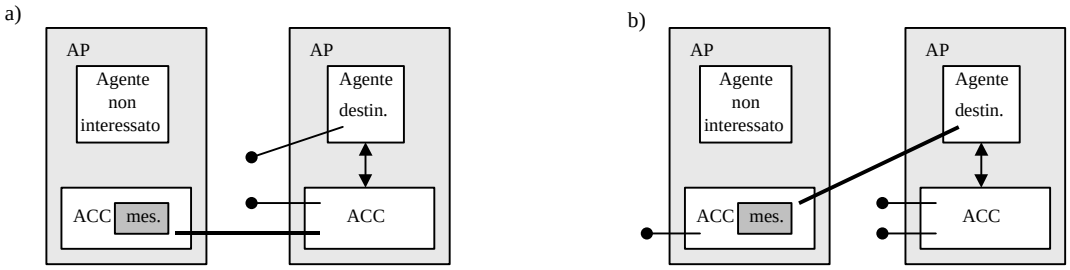


Fig. 20: l'ACC deve inviare il messaggio all'esterno della piattaforma. Nei due disegni il messaggio arriva a sua volta dall'esterno (pertanto gli agenti della sua piattaforma non sono interessati alla comunicazione), ma può anche arrivare da un agente locale (il mittente, ed in questo caso il messaggio non avrà *intended-receiver*). a) Il messaggio viene inviato ad un altro ACC (l'address dell'AID si riferisce ad un altro ACC), il quale proseguirà il processo di trasporto. b) l'ACC conosce l'indirizzo diretto di un agente con il quale condivide un sistema di trasporto ed invia il messaggio al destinatario senza altri passaggi intermedi. Per l'ACC non c'è differenza tra inviare un messaggio ad un'agente o ad un altro ACC, pertanto i casi a) e b) sono trattati in ugual modo

probabile) che questi si riferiscano ad un altro ACC, il quale si occuperà del passo successivo della trasmissione. Per procedere alla trasmissione l'ACC preleva il primo della sequenza ordinata di indirizzi presenti nell'AID, e tenta l'invio. Se va tutto bene, ha finito il suo compito. Può succedere però che il sistema di trasporto non sia disponibile, o che ci siano problemi di trasmissione. In questi casi, nell'envelope del messaggio (a questo punto non ancora inviato) viene aggiunto un parametro *intended-reicever*, contenente l'AID appena considerato, a cui però è stato tolto l'indirizzo verso cui la trasmissione è fallita. E di nuovo viene tentata la trasmissione verso il primo indirizzo dell'AID (e che prima era il secondo). Il processo continua fin quando non si riesce ad inviare un messaggio. Di volta in volta verranno aggiunti ulteriori parametri *intended-reicever*. Se la trasmissione non funziona con nessuno degli indirizzi presenti nell'AID, si ricorre agli eventuali resolvers indicati nell'AID. Il procedimento è lo stesso di quello seguito per gli indirizzi normali: si provano i risolutori in sequenza ordinata, e per ogni tentativo fallito si aggiunge un *intended-reicever* contenente l'AID privato del risolutore appena provato. È importante considrare che un risolutore può a sua volta avere più indirizzi. Quando risulta impossibile trovare un trasporto con il quale far proseguire il messaggio, esauritisi quindi sia gli indirizzi che i resolvers di questo, viene generato un messaggio di errore.

- se il messaggio è multicast l'ACC può, ma non è obbligato a farlo, decidere di separare il messaggio, in più copie ognuna con un sottoinsieme di destinatari del messaggio originale. Sceglierà il metodo più conveniente al caso: stando i destinatari su una stessa piattaforma, converrà non dividere i messaggi; stando i destinatari a gruppi su più piattaforme diverse, converrà mandare una copia per ogni piattaforma, ed ogni copia indirizzata ai destinatari presenti nella specifica piattaforma.

Come si vede in fig. 20, la consegna del messaggio al destinatario può avvenire sia dall'interno della piattaforma (fig. 20-a), che dall'esterno (fig. 20-b). La consegna dall'esterno richiede che l'ACC abbia come MTP almeno uno dei sistemi cui è collegato l'agente. La consegna interna è invece effettuata con il meccanismo fornito dall'AP. Anche quest'ultimo, comunque, può essere implementato attraverso un sistema "well-known", cioè standard, specialmente si la piattaforma si estende su più host.

Si termina la trattazione del servizio trasporto con la descrizione dei frame dell'ontologia FIPA-agent-management ad esso relativi, terminando i tal modo anche la trattazione di tale ontologia.

Si è visto che il parametro *received* è di tipo *received-object*. Questo costituisce un ulteriore frame che serve ad identificare un ACC per cui è passato il messaggio. I suoi parametri sono: *by*, contiene l'URL dell'ACC attuale (quello che ha appena ricevuto il messaggio), ed è obbligatorio; *from*, contiene l'URL dell'ACC che ha inviato il messaggio a quello attuale (non il mittente, bensì l'ultimo della catena degli ACC fino all'attuale), ed è opzionale; *date*, di tipo data/ora, indica l'istante di ricezione del messaggio da parte dell'ACC attuale, ed è obbligatorio; *id*, una stringa, identificante in maniera univoca il messaggio, opzionale; *via*, una stringa, identificante l'MTP sul quale si è svolta l'ultima trasmissione, opzionale. Beninteso, si è qui parlato di messaggi riferendosi all'elemento transport-message dell'architettura astratta, e non al FIPA-message.
Il frame *ap-transport-description* viene inserito dall'AMS all'interno degli *ap-description* (vedi 2.5.4), che gli vengono richiesti attraverso la funzione *get-description*. Lo scopo di *ap-transport-description* è quello di informare il richiedente riguardo tutti gli MTP con cui è collegato l'ACC della piattaforma. Questo frame dispone infatti un solo parametro: *available-mtps*, contenente una sequenza di *mtp-description*, uno per ogni MTP. Il frame *mpt-description* è a sua volta un descrittore del singolo MTP, e dispone dei seguenti parametri:

- *profile*: una stringa contenente il nome del *profilo trasporto* cui appartiene l'MTP considerato; opzionale;
- *mtp-name*: una stringa contenente il nome convenzionale del sistema di trasporto; opzionale;
- *addresses*: una sequenza di URL che rappresenta la lista di tutti gli indirizzi che l'ACC ha sul sistema di trasporto considerato (sul singolo sistema di trasporto ad un ACC può essere assegnato più di un indirizzo); obbligatorio.

Per promuovere l'interoperabilità, in sede FIPA si è deciso di stabilire degli insiemi minimi di MTP da supportare, chiamati *profili trasporto* (da non confondere con il mobile-agent-profile, 2.5.5). Per essere conforme FIPA, una piattaforma dovrà supportare almeno un profilo trasporto, il che vuol dire che dovrà collegarsi almeno a tutti gli MTP pervisti da tale profilo. Tali requisiti minimi di trasporto sono classificati in base agli ambienti in cui dovranno operare le piattafome concrete. Per ogni profilo trasporto è previsto un nome, una descrizione, ad un MTP di base.

2.6 – L'integrazione con software esterno

2.6.1 Un modello per l'integrazione con software esterno

Vari sono i motivi per cui bisogna ricorrere all'utilizzo di software non basato su agenti. Certamente la quantità di software esistente, e servizi da esso resi disponibili, non può essere semplicemente trascurata, pensando di risvilupare ogni componente software odierno con tecnologia ad agenti. È importante cioè la compatibilità con il passato. Senza un adeguato supporto per il software "non ad agenti" la diffusione del modello ad agenti stesso sarebbe destinata a non diffondersi semplicemente per mancanza di supporto (non sarebbe la prima volta, e non solo in informatica). Inoltre non è detto che il modello ad agenti si riveli vincente in ogni situazione. E non è detto che si diffonda, per così

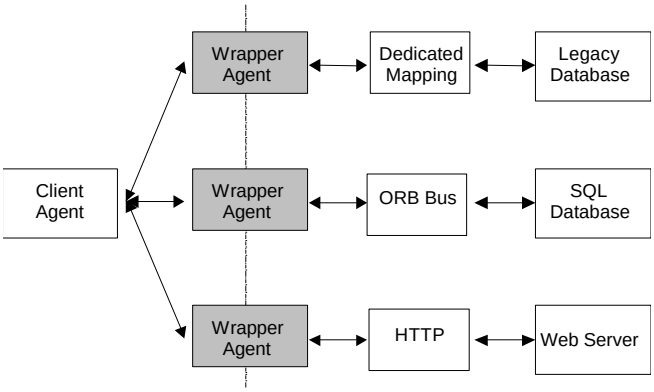


Fig. 21[20]: i wrapper devono interagire con il software esterno nei modi che questo prevede

dire, a livello globale, sostituendo dappertutto concezioni diverse del software. Ed in ultimo, anche a prescindere da queste ultime due eventualità, non è detto che non convenga realizzare un software esterno autocontenuto e farlo interagire poi con il "mondo degli agenti" attraverso il modello di integrazione che si cercherà di illustrare di seguito. Si vedrà infatti che lo scopo di questo modello è quello di rendere possibile la condivisione, tra la comunità di agenti, di risorse software esterne, nonchè il controllo in maniera dinamica di queste.

Un elemento fondamentale del modello è il *wrapper*. Si tratta di un agente che ha la facoltà di interfacciarsi in maniera propria con un elemento software esterno. Il compito del wrapper è quello di "girare" al software esterno le chiamate che gli arrivano da altri agenti, traducendo le richieste di operazione da messaggi ACL in chiamate di funzione o qualt'altro serva ad invocare le operazioni del software esterno. I wrapper costituiscono il ponte di passaggio tra il "dentro" ed il "fuori" del mondo degli agenti. Il modo in cui il wrapper si connette con il software esterno non è oggetto delle specifiche, poiché dipende da quali siano le interfacce esterne del software in questione. La fig. 21 mette in evidenza come un wrapper debba interfacciarsi con sistemi diversi a seconda del software cui deve accedere.

Niente vieta che un singolo agente possa svolgere le funzioni di wrapper per due distinti software. Nulla vieta inoltre ad un agente di potersi interfacciare con un software esterno per usufruirne solo in maniera personale, senza fare da wrapper per gli altri agenti. Più che un tipologia di agente, il wrapper è quindi una tipologia di servizio. Un agente può offrire il servizio di wrapper verso un certo software, e registrarsi presso il DF come fornitore di tale servizio: il suo service description dovrà avere il parametro type con valore *fipa-wrapper*. Ad un agente che svolga un servizio wrapper si richiede che sia capace di effettuare un collegamento dinamico con il software esterno: un wrapper non dovrà offrire l'interfaccia verso una singola istanza del software, ma potersi collegare dinamicamente a più istanze. Nel caso della figura precedente, deve, per esempio, poter sfruttare l'ORB per potersi collegare a diversi database SQL. È possibile (e molto conveeniente) che il wrapper supporti la possibilità di sostenere connessioni multiple con il software esterno.

In generale, per ogni tipologia di servizio dovrà essere prevista un'ontologia adatta sia nel caso di software fornitore interno al sistema degli agenti, sia in caso di software fornitore esterno. Il compito di "tradurre" le richieste di servizio (e quant'altro ad esso correlato) da messaggi ACL, con termini derivati da una certa ontologia, in qualcosa di comprensibile per il software esterno, è appunto del

wrapper. Ovviamente al wrapper è anche affidata la traduzione inversa nel caso contrario in cui sia il software esterno a dover informare il cliente di qualcosa.
L'altro elemento base del modello è l'ARB: Agent Resource Broker. Nel modello, ad ogni servizio software è associata una descrizione, che indica non solo le caratteristiche vere e proprie del servizio, ma anche dove e come è possibile raggiungere il software che lo rende disponibile. L'ARB è un broker di descrizioni di servizi: gli agenti normali, o i wrapper, possono registrare presso l'ARB le descrizioni dei servizi software che rendono disponibile, e gli agenti clienti possono ricercare, in base a specifici requisiti, il servizio cui sono interessati. Anche l'ARB, come il wrapper, è in realtà un servizio più che una categoria di agente. Il service description di un agente che, registrandosi presso il DF, dichiara di offrire il servizio di ARB, dovrà avere il parametro type con valore *fipa-arb*.

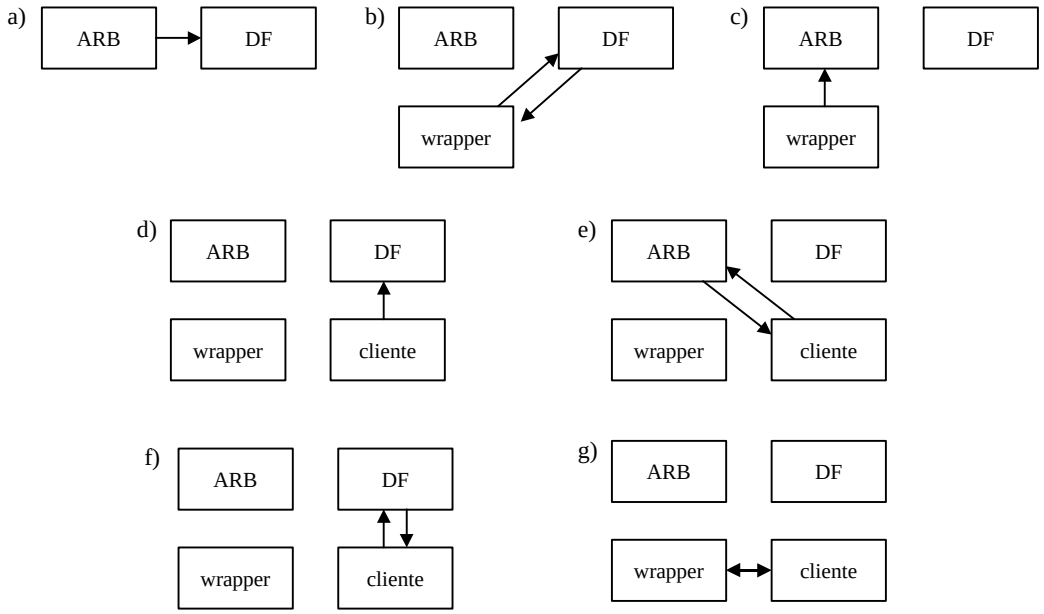


Fig. 22: Interazioni nel modello di integrazione con software esterno

In fig. 22 sono riportati tutti i passi affinchè un agente cliente possa cominciare effettivamente a servirsi di un servizio software esterno a partire da semplici requisiti. Prima di tutto, nel sistema deve esistere un ARB. Questo andrà (fig. 22-a) a registrarsi presso il DF come agente fornitore di un servizio di tipo *fipa-arb*. Il wrapper dovrà anch'esso registrarsi (fig. 22-b) presso il DF come agente fonitore di un servizio di tipo *fipa-wrapper*, e dovrà anche interrogare il DF su come trovare l'ARB. Dopodicchè, dovrà registrare (fig. 22-c) presso l'ARB la descrizione del servizio software esterno per cui fa, per l'appunto, da wrapper. A questo punto il wrapper è pronto per instaurare rapporti con gli agenti clienti. Un agente che vuole usufruire di un servizio software, dovrà innanzitutto (fig. 22-d) interrogare anch'esso il DF su come trovare l'ARB. Questo perché così potrà usare il servizio di brokering delle descrizioni dei servizi offertogli dall'ARB. Il cliente interrogherà (fig. 22-e) l'ARB mediante una specifica dei requisiti richiesti al servizio, e l'ARB gli ritornerà la descrizione del servizio esistente conforme ai requisiti richiesti. Il cliente potrà usare tale descrizione per interrogare (fig. 22-f) il DF sul come trovare il wrapper attinente. La risposta del DF permetterà (fig. 22-g) di localizzare il wrapper e poter cominciare così a sfruttare il servizio software.

Da quanto appena detto, segue che le descrizioni dei servizi presenti nell'ARB sono le stesse che i wrapper registrano nel DF. Nell'ARB sono registrati tutti i servizi software resi disponibili alla comunità di agenti: sia quelli interni alla comunità stessa, sia quelli esterni accessibili con l'intermediazione di un wrapper. Pertanto i passi visti possono attuarsi anche se il software fornitore del servizio è un agente. Da quest'ultima frase segue la completa integrazione tra mondo degli agenti e mondo esterno. Un agente, infatti, avendo a disposizione dei requisiti di suo interesse, attraverso l'ARB potrà venire a conoscenza del reale servizio disponibile concorde con tali requisiti, sia esso fornito da un agente o da un software esterno. Nel primo caso intratterrà il dialogo con l'agente fornitore, nel secondo con l'agente wrapper del software fornitore. Usando questo modello, un agente cliente può ignorare la differenza tra fornitore interno o esterno.
Sebbene il procedimento possa sembrare macchinoso, esso permette, inoltre, notevole flessibilità. Un cliente non è infatti obbligato a conoscere i dettagli del servizio che richiede, quindi la descrizione al completo, ma basta che specifichi all'ARB i requisiti richiesti al servizio. Per lo stesso motivo, se il software fornitore è esterno e se sono presenti più wrapper corrispondenti ai requisiti del cliente, questi potrà scegliere quale contattare (sempre attraverso il DF), magari in base al tipo di collegamento esistente tra il wrapper ed il software esterno.

2.6.2 Wrapper e ARB

Per scambiarsi messaggi riguardanti l'utilizzo di software fornitore, sono previste due ontologie FIPA: *FIPA-ARB* e *FIPA-Wrapper*.
FIPA-ARB dispone di tutti i frame necessari per esprimere concetti riguardanti le descrizioni dei servizi e di tutte le azioni necessarie per espletare le funzionalità dell'ARB.
service-description è il frame che che rappresenta la descrizione del servizio software. Dispone dei seguenti parametri:

- *name*: una stringa, contenente il nome convenzionale del servizio;
- *type*: una stringa, indicante la tipologia del servizio;
- *ontology*: un insieme di stringhe, indicanti le ontologie necessarie per trattare circa il servizio in questione;
- *protocol*: un insieme di stringhe, indicanti i protocolli necessari per trattare circa il servizio in questione;
- *properties*: un insieme di *property*, indicanti ulteriori caratteristiche del servizio (vedi 2.5.4)
- *communication-properties*: insieme di *communication-properties* (il frame ha lo stesso nome del parametro);

L'ultimo e i primi due parametri qui elencati sono obbligatori, gli altri sono opzionali.
Il frame *communication-properties* è il modo per esprimere come localizzare il software fornitore del servizio, in maniera indipendente dal servizio stesso. I suoi parametri sono:

- *net-protocol*: una stringa, indicante il protocollo di rete con cui è possibile connettersi al software; valori riservati sono per esempio SMTP HTTP, IIOP;
- *address*: un URL, contenente l'indirizzo di rete cui è possibile trovare il software;

- *message-body-format*: una stringa, indicante il formato del corpo dei messaggi scambiati con il software fornitore; un valore riservato di questo parametro è il *FIPA-String-ACL*, indicante il caso in cui il servizio software sia interno alla comunità degli agenti (cioè sia un agente);
- *message-body-encoding*: una stringa, indicante la codifica dei messaggi scambiati con il software fornitore.

Tali parametri sono tutti obbligatori.

Le function di cui dispone l'ontologia *FIPA-ARB* sono quelle supportate dall'ARB, che ha funzionalità simili a quelle dal DF: registrazione, modifica, cancellazione, ricerca.

- *register*: registra un servizio; l'argomento è un *service-description*; non ha range;
- *deregister*: cancella le informazioni relative ad un certo servizio; l'argomento è un *service-description*; non ha range;
- *modify*: modifica le impostazioni di registrazione di un certo servizio; l'argomento, un *service-description*, va a sostituire il *service-description* (con lo stesso parametro name) registrato in precedenza;
- *search*: l'argomento è un *service-description*; l'ARB usa l'argomento fornito come template per la ricerca nella sua base di conoscenza, il range è un set di *service-description*, cioè l'insieme di descrizioni corrispondenti al template.

FIPA-ARB dispone inoltre di un predicato di eccezione relativo al CA *failure*: *service-name-in-use*, riportata dall'ARB nel caso in cui, al momento della registrazione, sia già registrato un servizio con lo stesso nome (parametro *name*). L'argomento di tale eccezione è una stringa, contenente il nome in questione.

FIPA-Wrapper dispone dei frame e delle azioni necessari affinchè il cliente possa interfacciarsi con il wrapper, dando modo a quest'ultimo di "pilotare" il software esterno. A tale proposito è importante considerare il tipo di rapporto che il software esterno pretende di avere con i clienti, indipendentemente dal fatto che questi siano agenti o no. In altre parole, è importante il modello di comunicazione per cui un software è progettato: questi potrà infatti mettere a disposizione del cliente delle funzioni che alterino il suo stato interno, oppure potrà comunicare al cliente l'accadere di alcuni eventi, o altro. È importante che il modello di integrazione con software non basato su agenti disponga delle funzionalità necessarie per supportare tutti i possibili modelli di dialogo tra cliente e fornitore del servizio. I progettisti FIPA hanno diviso in tre categorie i possibili modelli di comunicazione cliente-fornitore:

- *event notification*: il fornitore notifica l'accadere di un certo evento agli agenti che si sono sottoscritti per quel particolare evento;
- *sensing functions*: il cliente richiede al fornitore di svolgere funzioni che non alterino lo stato del fornitore stesso, odell'ambiente, ma che restituiscano un valore di uscita, che il fornitore deve ritornare al cliente;
- *effecting actions*: l'agente chiede al fornitore di svolgere una azione, quindi un qualcosa che alteri il suo (o quello di qualcos'altro) stato interno.

L'agente che si interfaccia con il software esterno è il wrapper, che dovrà pertanto sopportare operazioni con le quali gestire tutte le categorie di modelli di comunicazione cliente-fornitore. Le operazioni che devono essere supportate dal wrapper sono dunque:

- richiedere connessioni dinamiche con il software esterno; requisito essenziale di un wrapper;

- interrogare il software esterno sulle sue proprietà ed impostazioni; requisito essenziale di un wrapper;
- impostare parametri del software esterno; requisito essenziale di un wrapper;
- invocare operazioni sul software esterno; indispensabile per il supporto il modello *effecting actions*;
- essere informato circa il risultato di una funzione; indispensabile per il supporto il modello *sensing functions*;
- sottoscriversi a particolari eventi del software esterno; indispensabile per il supporto il modello *event notification;*
- terminare le connessioni dinamiche con il software esterno; requisito essenziale di un wrapper.

L'ontologia di comunicazione tra cliente vero e proprio e wrapper dovrà disporre dei termini necessari per supportare anch'essa tutte le categorie di modelli di comunicazione cliente-fornitore, e delle azioni con le quali attivare le operazioni di un wrapper appena viste. Tale ontologia è *FIPA-Wrapper*.

I frame supportati da questa sono gli stessi di *FIPA-ARB*: *service-description* e *communication-properties*.

Le function supportate da *FIPA-Wrapper* sono:

- *init* e *close*: inizializza e termina, rispettivamente, il sistema software esterno, relativamente alla singola connessione. L'inizializzazione ha come dominio un *service-description*, il servizio software cui si vuole accedere, e un set di *property*, che servono come parametri di inizializzazione del software. L'inizializzazione ha come range un *service-instance-id*, che identifica la singola richiesta di servizio (connessione) con il software esterno. La terminazione ha come dominio il *service-instance-id* relativo alla connessione da chiudere, seguito da un set di *property*, indicanti parametri aggiuntivi riguardanti la terminazione stessa. Non ha range.
- *store* e *restore*: la prima permette di ottenere lo stato del sistema software relativamente alla singola connessione, identificata questa nel dominio da un *service-instance-id*. Il range è uno *state-id*, che per l'appunto identifica lo stato del sistema software nell'istante in cui è chiamata la *store*. La seconda ripristina lo stato di un sistema software in base alle informazioni ottenute in precedenza dalla *store*. Il suo dominio è composto da un *service-instance-id*, relativo alla connessione cui si riferisce lo stato interno che verrà ripristinato, e da uno *state-id*[10], identificante lo stato da ripristinare. Non ha range.
- *suspend* e *resume*: in alcuni casi un cliente può chiedere la sospensione e la successiva riattivazione del servizio software, relativamente alla singola connessione. Entrambe le funzioni hanno come dominio la connessione per cui il servizio deve essere sospeso o riabilitato, cioè hanno come dominio un *service-description-id*. Entrambe le funzioni non hanno range.
- *software-subscribe* e *software-unsubscribe*: nell'ambito dei servizi realizzati con modello di comunicazione di tipo event notification, il cliente può sottoscriversi a certi eventi, o cancellare una sua sottoscrizione precedente. Entrambe le funzioni hanno come dominio il *service-description-id* della connessione su cui è fatta la richiesta, e un *event-name*, cioè l'identificatore dell'evento cui ci si vuole sottoscrivere. Entrambe le funzioni non hanno range.

- *invoke*: nell'ambito dei servizi realizzati con modello di comunicazione di tipo effecting actions, il cliente può richiedere al sistema software lo svolgimento di alcune azioni. L'ontologia FIPA-Wrapper indica con il nome *functional-expression* la generica azione richiesta al sistema esterno. L'operazione *invoke* pertanto avrà come dominio il *service-description-id* relativo alla onnessione su cui viene effettuata la richiesta, e il *functional-expression* indicante la precisa azione da compiere. La funzione non ha range.

2.7 – Considerazioni sul modello FIPA

2.7.1 Architettura astratta + modello di riferimento: flessibilità e potenza

Si faranno alcune considerazioni sul modello illustrato, puntando su certi aspetti intrinseci in quanto finora detto, ma che probabilmente è bene esplicitare.
Si è visto come l'architettura astratta, che traccia linee guida estremamente generali ed applicabili in qualunque ambiente, ed il modello di riferimento, che estende e si sviluppa sopra l'architettura astratta, costituiscano i punti principali delle specifiche FIPA 2000. Si sono visti i rapporti i due. Si è visto come il modello FIPA consenta l'interoperabilità tra due agenti, al di là del limite imposto dalla diversità degli ambienti su cui i due agenti possono operare.
Ma oltre ciò, un pregio, sicuramente frutto di sforzo in sede FIPA, del modello illustrato è quello di permettere anche notevole flessibilità agli agenti, i quali sono sempre nella possibilità di scegliere a quale "livello" operare relativamente ad un certo aspetto delle loro funzionalità. Si è visto come un agente possa essere collegato, in proprio o attraverso l'ACC della piattaforma sulla quale opera, a più sistemi di trasporto. Il modello permette ad un agente che voglia comunicare, di gestire la localizzazione dell'interlocutore con una politica di alto livello, così da non doversi preoccupare di scegliere un certo trasporto piuttosto che un altro lasciando tale compito all'MTS, a tutto vantaggio della semplicità e della manegevolezza del codice. Viceversa, ad un agente è data la possibilità di scegliere con quale trasporto contattare l'interlocutore, e di gestire i dettagli della comunicazione (tipo di servizio, multicasting, con o senza connessione), a tutto vantaggio dell'efficienza.
Ciò vale per ogni funzionalità dell'agente. Per esempio, ad un agente è offerta la possibilità di basare dialoghi ACL sui protocolli di comunicazione, in modo tale da non doversi preoccupare oltre al fine di arrivare ad una conversazione di senso compiuto. E viceversa gli è offerta la possibilità di strutturare dinamicamente i dialoghi nel caso non siano previsti protocolli validi per il tipo di comunicazione da compiere. Quest'ultimo approccio alla conversazione è sicuramente di livello più basso, poiché maggiori sono i dettagli cui bisogna prestar attenzione.
La flessibilità a disposizione dello sviluppatore di agenti sta anche nel fatto che questi è libero di implementare agenti che gestiscano da se le code dei messaggi ordinate per conversazione, oppure

[10] Al momento della scrittura, le specifiche relative all'integrazione del software nelle piattaforme ad agenti non includono quest'argomento nel dominio della funzione *restore*. Ciononostante, per lo scopo della funzione considerata, la sua presenza sembrerebbe ovvia.

che si basino sul supporto della piattaforma, oppure ancora che semplicemente non gesticano conversazioni contemporanee.
La disponibilità di multipli sistemi di codifica del FIPA-message e del transport-message può rendere estremamente efficiente la comunicazione all'interno dell'ambiente in cui opera un agente, e con il quale si presume che questi abbia i maggiori scambi di messaggi, lasciando la possibilità di effettuare dialoghi, per così dire, a distanza (magari tra mondi completamente differenti come quelli di un laptop e di un mainframe) mediante l'uso di codifiche comuni.
Esiste la possibilità per un agente di rimanere anonimo, di gestire comunicazioni con e senza connessione, o magari utilizzare il "durable messaging", con il quale l'agente invia il messaggio comunicando al sottostante MTS che la consegna dovrà essere effettuata in seguito, in un momento stabilito.
Il discorso appena fatto non vale invece per il DF e la richiesta di informazioni su altri agenti. L'approccio qui è stato diverso. L'architettura astratta, e così il modello di riferimento, prevede che il directory-service si basi su più sottostanti sistemi di gestione directory distribuite. L'architettura astratta permette in effetti all'agente di attuare a questo riguardo una politica di alto livello, lasciando alla AP il compito di scegliere il sistema directory da utilizzare, o di basso livello, controllando da sé gli aspetti specifici di tale sistema. Ciononostante, il modello di riferimento, con l'introduzione del DF, elimina quest'ultima possibilità. Questo perchè comunque all'agente è ora permesso di compiere ricerche più sofisticate e, se bisogna, precise. Il DF si accollerà il compito di gestire per l'agente i compiti di interrogazione dei vari sistemi directory utilizzati dalla AP, nonché di cooperare con altri DF della piattaforma o esterni, allo scopo di reperire le informazioni cercate.
La libertà lasciata allo sviluppatore del singolo agente è notevole. Ma le potenzialità a disposizione dello sviluppatore dipendono comunque parecchio dalla base hardware/software e da quelle che sono le potenzialità della piattaforma concreta su cui opererà l'agente. Il primo fattore è di per sé ovvio: come ogni processo, anche un agente risente delle caratteristiche del linguaggio in cui è sviluppato, del sistema operativo sul quale gira, dell'hardware a disposizione. Il secondo fattore è ciò che differenzia il modello FIPA di piattaforma da una piattaforma reale. Le API fornite a livello di agente e quant'altro fornito dalla piattaforma come strumento di sviluppo saranno di importanza fondamentale per lo sviluppatore. Funzionalità quali compressione dei messaggi, crittografia dei messaggi, parser della sintassi ACL, eventuali parser XML o HTTP, codifica/decodifica MIME, supporti per varie tipologie di codifiche dei caratteri e framework ad oggetti costituiranno punti a favore per certe piattaforme piuttosto che per altre. Sarebbe inutile dirlo: la costruzione di un agente è più complessa della costruzione di un generico processo, in gran parte perché è più complesso il modello di linguaggio che l'agente adopera, e cioè il modello costituito dall'unione dei linguaggi semantici con le ontologie.

2.7.2 Mancata comunicazione

Si vuole ora elencare le ragioni di una eventuale impossibilità di comunicazione tra due agenti. Le motivazioni sono inserite in ordine di livello di comunicazione crescente (i livelli di comunicazione sono stati introdotti in 1.2.3):

- le piattaforme sulle quali operano i due agenti non hanno *sistemi di trasporto comuni*, e non esiste un percorso attraverso altre piattaforme, e quindi *gateway tra un sistema di trasporto e*

l'altro, che possa portare alla comunicazione delle due piattaforme; rarissimo (per non dire impossibile): l'idea dei profili trasporto (2.5.8) nasce apposta per evitare questa situazione, ed in ogni caso il numero di sistemi trasporto non è talmente grande che possa non essere possibile, attraverso vari ACC, la conversione dei messaggi da un trasporto ad un altro;

- non esiste un metodo di *codifica dell'envelope* in comune tra i due agenti; molto raro: esistono codifiche dell'envelope magari non troppo efficienti, ma sicuramente abbastanza semplici da poter essere implementate dovunque;
- non esiste un metodo di *codifica del payload* in comune tra i due agenti; raro: anche in questo caso esistono buone possibilità che si trovi una codifica in comune perché ne esistono di parecchio semplici (per esempio quella in stringhe); meno raro del caso precedente perché ci sono più possibilità a disposizione;
- non esiste un *protocollo di interazione* in comune tra i due agenti (relativamente al tipo dia conversazione da effettuare); possibile: dato il tipo di conversazione da effettuare (richiedere azioni, dare informazioni, fare domande), è possibile che non ci si intendda sui protocolli, ma non troppo probabile;
- non esiste un *content language* in comune tra i due agenti; probabile: i content language sono scelti dallo sviluppatore dell'agente per la loro capacità di adattarsi al modello di conoscenza che lo sviluppatore stesso deve rappresentare nell'agente, e che quest'ultimo dovrà scambiare con altri agenti. I content language scelti dipendono insomma dalla tipologia di applicazione cui si presta l'agente, e pertanto esiste la possibilità che non siano in comune con agenti con cui non si condivide tale tipologia; resta comunque il fatto che i content language disponibili al momento sono solo quattro;
- non esistono *ontologie* in comune tra i due agenti; probabilissimo: esistono ontologie per ogni campo della conoscenza, per cui un agente potrà colloquiare con altri agenti soltanto riguardo lo specifico settore applicativo (quindi con pochi agenti, relativamente a tutti quelli presenti nell'ambiente), cioè con agenti con cui condivide ontologie.

A questi si potrebbe pensare di aggiungere il mancato intendimento di un CA, ma sarebbe un errore perché questo si riferisce al singolo messaggio, non alla possibilità generica di comunicazione tra due agenti.

Si è fatto questo elenco anche per precisare ulteriormente quali siano i livelli di comunicazione (in corsivo) che entrano in gioco durante lo scambio messaggi tra un agente ed un altro nel modello FIPA.

2.7.3 Sicurezza nelle piattaforme FIPA

La sicurezza è uno di quegli argomenti su cui non si è riusciti a trovare un modello standard da inserire nelle specifiche. La versione 98 delle specifiche FIPA conteneva un documento riguardante la sicurezza, ma è stato abbandonato nelle nuove specifiche 2000, nelle quali l'argomento risulta di nuovo come non trattato, aperto quindi alla discussione. E soprattutto non standardizzato, per cui il rischio è quello di una eccessiva diversificazione nelle implementazioni reali delle tecniche relative alla sicurezza, tale da ostacolare future standardizzazioni.

Il problema è che i modi ed i meccanismi con cui implementare la sicurezza, nonché le aree che tali meccanismi devono coprire, sono normalmente dipendenti dal settore applicativo cui è dedicato un

sistema. In più, la complicazione è che su una particolare piattaforma possono operare agenti con funzionalità differenti, magari operanti nell'ambito di sistemi diversi. Questo fa sì che neanche la scelta a livello di implementazione sia facile.

Per riparare, almeno in parte, a tutto questo, nel documento di specifica dell'architettura astratta vengono descritte in maniera informale delle linee guida sull'implementazione delle tecniche di base della sicurezza.

Gli aspetti della sicurezza che interessano direttamente il modello ad agenti sono:

- *identità*: un agente può voler riconoscere l'identità di un altro agente con cui interagisce, per decidere quale politica adottare; ovviamente il semplice sistma dell'AID nel campo sender del messaggio ACL mal si presta ad implementazioni con elevato grado di sicurezza, poiché risulta possibile che il mittente di un messaggio inserisca nel campo suddetto (ma anche in altri) informazioni appositamente errate, allo scopo di presentarsi con l'identità di un altro agente; l'identità inoltre può fare riferimento non solo all'agente in se stesso, ma anche al suo attuale proprietario, sia esso una persona fisica oppure una società o ente generico; conoscendo l'identità dell'interlocutore un agente può essere sicuro di potere inviargli certe informazioni o decidere di affidarsi ai suoi servizi o di effettuare transazioni legate proprio all'identità dell'interlocutore (tipico esempio: pagamenti con carta di credito); l'identità deve essere gestita con tecniche di firma digitale ed autorità di certificazione: elementi che aumentano la complessità del sistema o della piattaforma;
- *permessi di accesso*: un agente, per quanto detto al punto precedente, in base all'identità dell'interlocutore, può decidere quali risorse rendergli disponibili, quali azioni rendere possibili, o altro che possa limitare le funzionalità nei confronti di interlocutori senza sufficienti permessi;
- *validità*: può esserci il bisogno di sapere se un certo elemento è stato modificato da terzi; per esempio se un messaggio sia stato modificato durante il tragitto seguito per arrivare a destinazione, oppure se il codice di un agente sia stato modificato allo scopo di compiere azioni illecite approfittando dell'identità associata all'agente stesso, oppure ancora se il codice di un agente sia stato modificato durante una migrazione; anche per la validità solitamente si ricorre a tecniche di firma digitale;
- *riservatezza*: può esserci il bisogno di impedire a terzi di ricevere un messaggio scambiato tra due agenti; per ottenere questo il messaggio deve essere crittografato, oppure bisogna fare ricorso a canali di trasmissione sicuri.

Relativamente all'ultimo punto si vuole fare la distinzione tra messaggio crittografato e canale sicuro: il primo implica che sia l'agente a preoccuparsi della crittografia dei dati che incanalerà sul mezzo di trasporto; il secondo invece presuppone che il mezzo di trasporto implementi già di per se stesso sistemi di crittografia, ed in tal modo il mittente non deve preoccuparsi di gestire internamente questioni ad essa relative. Esempi di tali canali di comunicazione sono l'IPSEC, il CORBA Common Secure Interoperability Service, il Secure Socket Layer.

Per l'implementazione delle possibilità espresse nei punti precedenti il meccanismo cui si fa comunemente ricorso è quello dei sistemi crittografici a chiave simmetrica unitamente a servizi di autorità di certificazione. Un modello che faccia uso di tali mezzi presuppone che ogni agente possegga una chiave privata ed una relativa chiave pubblica comunicata all'autorità di certificazione. La chiave privata può essere usata per firmare i messaggi in uscita dall'agente (firma digitale). Il ricevente potrà verificare se l'identità contenuta nel messaggio è esatta riconoscendo la firma

digitale mediante la chiave pubblica richiesta in precedenza all'autorità di certificazione. Inoltre la chiave pubblica del destinatario può essere usata del mittente per crittografare il messaggio, in maniera tale che questi possa essere decrittigrafato solo dal destinatario per mezzo della sua chiave privata.

Questo meccanismo prevede però che la chiave pubblica sia certificata, cioè che si sappia realmente a chi appartenga. Viceversa il sistema continua ad essere insicuro poiché può esserci associazione di un agente con la chiave pubblica di un altro.

Sebbene questi siano i settori più importanti su cui verte l'analisi della sicurezza di un una piattaforma ad agenti, esistono altre problematiche relative all'argomento per le quali non risulta possibile trovare soluzione, sia per limiti intrinseci del modello, sia perché molto dipende dai singoli agenti componenti il sistema. Limiti intrinseci del momdello sono per esempio la possibilità di attacchi contemporanei ad un agente allo scopo di provocare un *denial of service*, cioè l'invio contemporaneo da più parti di messaggi senza significato, causando lo spreco di risorse (banda disponibile, carico sul tempo macchina,…) dell'agente attaccato al solo scopo di impedirne la funzionalità, oppure la possibilità che si verifichino le cosiddette *misinformation campaigns*, campagna di disinformazione, che consiste nella diffusione di notizie false circa la presunta pericolosità (o inefficienza, o insicurezza, o costosità,…) di un agente che perciò viene "emarginato", o ripudiato, nei rapporti con gli altri. Caratteristiche riguardanti la sicurezza dipendenti dai singoli agenti si manifestano invece durante tentativi di *probing*, o *data peeping*: in questo caso vengono utilizzate funzionalità regolari di un agente per estrarre da questo informazioni utili per scopi non regolari. Un agente sprovveduto, ed in un certo qual modo "ingenuo", potrebbe avere la possibilità di fornire troppe informazioni all'esterno senza rendersi conto che qualcuno potrebbe apprifittarne.

Problemi relativi alla sicurezza esistenti in quasi tutti i sistemi infomatici sono inoltre lo *spoofing* ed il *masquerading*: una entità maligna può riuscire ad appropiarsi dell'indirizzo di un'altra (spoofing) e sostituirsi ad essa nello scambio di informazioni come se nulla fosse accaduto; per far questo si può approfittare di problemi, per esempio crash, avuti dall'entità cui ci si vuole sostituire (masquerading), magari provocati appositamente. Un fenomeno simile allo spoofing può esistere anche in un sistema con crittografia a chiave simmetrica se il servizio di certificazione non è progettato correttamente. Un agente maligno può, in questo caso, spacciarsi per un altro al momento della consegna della chiave pubblica al servizio di certificazione. Così facendo, l'agente viene a tutti

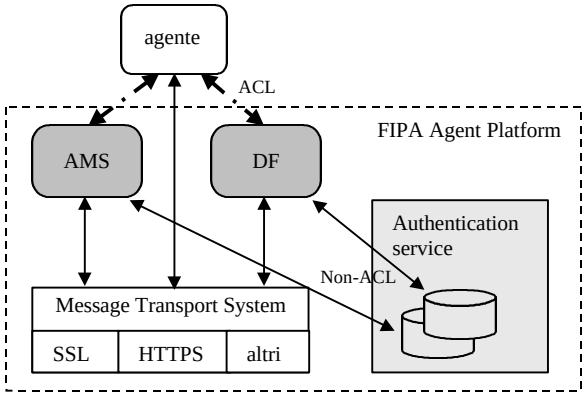


Fig. 23 [8]: un possibile modello per l'implementazione della sicurezza in una piattaforma FIPA

gli effetti identificato come avente certe funzionalità, e certi requisiti di sicurezza, che invece non ha. In fig. 23 è schematizzato uno dei modelli possibili per l'implementazione della sicurezza in una piattaforma FIPA ([8]). Questo modello prevede che:

- l'AMS ed il DF identifichino, attraverso firma digitale, gli agenti che fanno richieste, per esempio di registrazione o modifica, prevenendo così il masquerading;
- l'agente identifichi, attraverso firma digitale, l'AMS ed il DF quando effettua qualche richiesta, in modo da non rischiare di scambiare informazioni importanti con estranei;
- si utilizzino canali di trasporto sicuri, con crittografia a chiave simmetrica: il servizio di autenticazione lega il nome dell'agente con la sua chiave pubblica.

Le comunicazioni con l'authentication service dovranno svolgersi anch'essi su canali privati se si vogliono evitare la sostituzione della chiave pubblica nella trasmissione dall'authentication service stesso all'agente richiedente, o addirittura spoofing dell'authentication service.
Nel modello non sono però definiti il meccanismo di gestione (creazione, distribuzione) e di certificazione delle coppie di chiavi. Argomenti, come si è visto, di importanza fondamentale se si vuole aver fiducia in questi meccanismi di sicurezza.

2.7.4 Precisazioni sugli agenti mobili

Chiarire le differenze tra concetti diversi, ma interrelati, indicati con termini uguali, può sicuramente essere utile.
Con il concetto di codice mobile si indica un generico programma in grado, attraverso piattaforme di supporto, di spostarsi da una piattaforma ad un'altra, compatibile con il suo tipo di codice. Il concetto è di per sé scorrelato dal concetto di agente con cui finora si è avuto a che fare. Ma il bisogno di piattaforme di supporto e, per l'appunto, la capacità di vagare da un punto ad un altro della rete, mantenendo lo stato interno, per svolgere compiti di qualsiasi genere per conto dell'utente proprietario, ha fatto si che per programmi sviluppati con codice mobile si cominciasse ad usare il termine "agente mobile". Questo, di nuovo, senza nulla avere a che fare con linguaggi semantici, ACL e interoperabilità. A questo bisogna aggiungere che gli agenti come li abbiamo finora intesi possono anch'essi, nei casi e nelle modalità viste, essere mobili.
Ecco quindi che si può generare confusione nell'uso dei termini agente e agente mobile. Per distinguere il concetto di agente come l'abbiamo finora visto ed agente mobile, intendendo con questo termine un programma a codice mobile, si indicherà il primo con il termine di agente intelligente. Distinguiamo quindi tra: agenti mobili, agenti intelligenti ed agenti intelligenti mobili.
Anche per le piattaforme di supporto per agenti mobili si è cercato di arrivare ad una standardizzazione. L'OMG, a tal proposito, ha proposto un insieme di specifiche atte allo sviluppo di piattaforme per agenti mobili basate su CORBA e sul protocollo IIOP, entrambi standardizzati dalla stessa OMG. Tali specifiche prendono il nome di MAF, Mobile Agent Facilitator.
Data la differenza di cui ora si è detto, e dato che esistono, a questo punto, le prerogative per la diffusione del modello ad agenti intelligenti, si cerca di convertire le piattaforme per agenti mobili finora sviluppate in piattaforme per agenti mobili intelligenti. Questo vuol dire cambiare non solo la pattaforme in se stessa, ma anche tutti gli agenti che vi operano di sopra, poiché devono essere messi in grado di operare con l'ACL e con le ontologie.

Parte 3 – Conclusioni

L'idea di rete globale ha avuto, ed avrà ancora, un impatto sull'uomo maggiore di quanto ci si aspettava al momento del suo concepimento. L'idea è quella di permettere la comunicazione tra due qualsiasi punti del globo. Lo scambio di informazione è quindi alla base del fenomeno Internet che sta modificando il modo di vivere e di lavorare, l'economia e l'impresa, la ricerca e l'istruzione.
Il modello ad agenti si basa sulla rete globale, ed aggiunge ad essa la capacità dell'interoperabilità. Con l'utilizzo di questo modello, diviene possibile l'interoperabilità funzionale tra due entità software funzionanti non solo in posti diversi, ma soprattutto in ambiti diversi ed in ambienti diversi. In tal modo è possibile costituire un unico sistema con l'interoperabilità di più sottosistemi.
L'agente rappresenta la prima concezione di software applicativo contenente delle caratteristiche di intelligenza artificiale (da questa derivano i concetti di linguaggio semantico e di ontologia). E l'ACL è il concetto essenziale di distinzione tra un agente ed un processo qualunque. La comunicazione ad un livello semantico, il potere esprimere conoscenza, è la caratteristica innovativa, che rende possibile l'astrazione al punto da permettere lo scambio di servizi in maniera indipendente dall'ambito in cui i servizi stessi risultano disponibili. Si superano così differenze legate alla base hardware e software sulle quali operano gli agenti, e, almeno teoricamente, risulta possibile superare le differenze linguistiche (si ricorda che un'ontologia rappresenta le cose in sé, e non il modo di riferirvisi).
Difficilmente altre invenzioni (termine usato in maniera abbastanza impropria) potranno avere un influsso sulla società come quello che sta avendo l'internet. Resta però il fatto che non è il concetto di Internet ad influire sull'uomo, ma i modelli di comunicazione che attraverso di questa ci vengono proposti: world wide web, posta elettronica, solo per indicare più diffusi. É su questi che si sviluppano le realtà applicative. Per esempio: e-commerce, business-to-business, pubblicità e marketing, svago, nel caso del web; organizzazione individuale ed aziendale, rapporti sociali, nel caso della posta elettronica. Se il modello ad agenti avrà successo, avrà dunque anch'esso notevole impatto sull'uomo.
Non è questa l'occasione giusta per fare previsioni sul futuro del software ad agenti, e non è neanche il momento giusto. Ci sono troppe variabili, di cui molte non deterministiche.
Spesso gli sviluppatori hanno frenato la diffusione di nuovi concetti frutto della ricerca. Un esempio è dato dal fatto che molti programmatori, e per molti anni, hanno tenuto un atteggiamento di rifiuto riguardo alle metodologie object-oriented per il semplice fatto che non riusciveno a vedere le motivazioni per investire tempo e denaro nell'aggiornamento ad una nuova tecnologia, che per di più cambia in maniera abbastanza radicale la visione di un certo sistema rispetto a tecnologie precedenti (p.es. programmazione funzionale strutturata e progettazione SSADM). Questo fenomeno è

destinato ad essere maggiorato nel caso degli agenti. La scrittura di un agente è sicuramente più difficile che non la scrittura di un semplice processo (a parità di compiti svolti), pur anche utilizzando appositi tool di sviluppo, ed inoltre richiede maggiori conoscenze.
D'altra parte l'industria ha spesso ignorato nuovi concetti frutto della ricerca, ma ciò avviene solo fino a che non balzano agli occhi le possibilità applicative, o in generale l'utilità (con il doppio senso di essere utile e di dare utile, cioè guadagno). E la potenza del modello ad agenti intelligenti è sicuramente notevole dal punto di vista applicativo, oltre al fatto che il primo stadio di ricerca su questi argomenti è concluso, ed è possibile cominciare quello di produzione.
In effetti le specifiche FIPA giungono nel momento opportuno. Non troppo presto, così che ci sia già una adeguata conoscenza dell'argomento (acquisita nel corso della prima fase di ricerca) e sia improbabile che nuove conoscenze possano portare allo sconvolgimento di quelle attuali. Non troppo tardi, così che siano parecchie le chance di dirigere e standardizzare realmente il mercato che va creandosi, anzicchè tentare di standardizzare un mercato oramai variegato e costellato di implementazioni non compatibili tra loro ([7]). FIPA è un consorzio industriale, il che vuol dire che l'industria sta favorendo la diffusione del modello ad agenti.
Va anche considerato che la forza di un modello come quello ad agenti, atto alla interoperabilità, sta nel supporto che gli viene dato: maggiore è il numero di agenti a disposizione, maggiore è l'interoperabilità. Per cui (ma non è solo il caso del modello ad agenti) maggiori saranno i rischi dei "pionieri" della nuova tecnologia, maggiore sarà lo stimolo a continuare per questa strada.
Ma queste sono tutte cose che solo il tempo potrà mostrare.

Bibliografia

[1] Buckle, P. e Hadingham, R. “FIPA and the Internet Revolution”, Nortel Networks, 1999

[2] Comer, D. “Internetworking with TCP/IP”, Prentice-Hall

[3] Guarino, N. “Formal ontology, conceptual analisys and knowledge representation”, International Journal of Human-Computer Studies, 5(6):625-640, 1995

[4] Meyer, B. “La produzione del software object-oriented”, Prentice-Hall, 1988

[5] Nwana, H. “Software Agents: An Overview”, Knowledge Engineering Review, 11(3):205-244, 1996

[6] O’Brien, P.D. e Nicol, R.C. “FIPA– towards a standard for software agents”, BT Technology Journal, 16(3):51-59, 1998

[7] Poslad, S. Buckle, P. e Hadingham, R. “Open Source, Standards and Scaleable Agencies”, Proceedings of PAAM 2000, Manchester UK, 2000

[8] Poslad, S. e Calisti, M. “Towards improved trust and security in FIPA agent platforms”,

[9] Searle, J.R. “Speech Acts”, Cambridge University Press, Cambridge UK, 1969

[10] Shoham, Y. “An overview of agent-oriented programming languages”, 16(8):57-67, IEEE Computer ,1997

[11] Thissen, R.L. “Gadget, a generic FIPA-compliant agent development toolkit”, Master Thesis, Universiteit Maastricht, 1999

[12] Tanenbaum, A.S. “I moderni sistemi operativi”, Prentice-Hall 1992

[13] “FIPA00001 – FIPA Abstract Architecture Specification”, Fondation for Intelligent Phisical Agents, 2000

[14] “FIPA00023 – FIPA Agent Management Specification”, Fondation for Intelligent Phisical Agents, 2000

[15] “FIPA00087 – FIPA Agent Management Support for Mobility Specification”, Fondation for Intelligent Phisical Agents, 2000

[16] “FIPA00067 – FIPA Agent Message Transport Message Specification”, Fondation for Intelligent Phisical Agents, 2000

[17] “FIPA00061 – FIPA ACL Message Structure Specification”, Fondation for Intelligent Phisical Agents, 2000

[18] “FIPA00007 – FIPA Content Language Labrary Specification”, Fondation for Intelligent Phisical Agents, 2000

[19] “FIPA00086 – FIPA Ontology Service Specification”, Fondation for Intelligent Phisical Agents, 2000

[20] “FIPA00079 – FIPA Agent Software Integration Specification”, Fondation for Intelligent Phisical Agents, 2000