Andrea Novaga

Progetto del corso di Sistemi Multi-Agente LS

Test e Valutazione di SODA e delle metodologie Agent-Oriented

Sommario

Oggetto di studio della presente relazione è l'attuale situazione delle metodologie per lo sviluppo di software orientato agli agenti: si effettuerà una panoramica generale delle metodologie fino ad oggi sviluppate, andando poi ad analizzare più nel dettaglio alcune fra quelle più importanti e utilizzate (MaSE, Prometheus, Tropos, Gaia) ed un caso di metodologia relativamente più giovane (SODA), e si analizzerà fino a che punto siano state testate e messe a punto fino ad ora; quindi si motiverà il perché dell'importanza di trovare dei metodi di valutazione e confronto fra queste metodologie e si riporteranno esempi di lavori svolto da diversi ricercatori in quest'ambito, dei criteri di valutazione trovati, dei relativi framework realizzati e dei risultati così conseguiti. Tutto quanto riportato è stato tratto e ricavato dai resoconti di altre ricerche, da parte di ricercatori diversi, riportati in bibliografia.

Indice

1. Introduzione

L'evoluzione del software negli ultimi anni, software che è divenuto sempre più complesso, distribuito e pervasivo, ha determinato il successo e la veloce affermazione dell'approccio orientato agli agenti, il quale ha praticamente sempre dimostrato di ben adattarsi a questo nuovo scenario. Si è dunque verificata una rapida diffusione e diversificazione di modelli, infrastrutture e linguaggi orientati agli agenti, o che comunque si avvalgono di concetti legati al paradigma d'agente. A tale fenomeno però non è seguita una altrettanto rapida evoluzione tecnologica per supportare tali astrazioni, dal momento che piattaforme e applicazioni specifiche per lo sviluppo di sistemi multi-agente, o di sistemi software complessi fondati sul concetto di agente, sono per lo più ancora in fase di sviluppo. Il modello agent-oriented è d'altronde ancora in una fase di transizione dalla prototipizzazione, da parte dei ricercatori, allo sviluppo industriale. Quel che si ha dunque allo stato attuale è un gap fra modelli / infrastrutture / linguaggi e le tecnologie di supporto. In questo contesto diventa di fondamentale importanza l'approccio metodologico, come supporto di base per agevolare gli ingegneri del software nel disegno e nello sviluppo di sistemi complessi, fondati sul paradigma agent-oriented.
Numerose nuove metodologie, basate su tale paradigma, sono allora state ideate e ulteriormente perfezionate, generalmente a partire da altre metodologie esistenti, fondate o sempre sul concetto di agenti, oppure sugli altri concetti preesistenti (in particolare quello di oggetti). Il panorama di questa nuova branca della software engineering, sembra dunque offrire una ampia scelta fra le diverse metodologie per il processo di sviluppo software orientato agli agenti, tuttavia di tali metodologie molte sono ancora nelle prime fasi di definizione o sviluppo e richiedono ancora che ne sia verificata l'effettiva adeguatezza. Non tutte si mostrano infatti egualmente efficaci, spesso il problema è dovuto al fatto che esse si basano su modelli “vecchi”, non legati al concetto di agente ed estesi/ridefiniti in seguito allo scopo. Alla fine comunque, anche dopo che molte di queste metodologie sono state testate e verificate attraverso diversi casi di studio, ci rimane una scelta non indifferente fra quelle disponibili; si tratta oltretutto di metodologie assai differenti fra loro nel complesso, che affrontano il problema del processo di sviluppo secondo ottiche e approcci differenti, quindi lo studio approfondito di una difficilmente può portare ad una comprensione generalmente accurata di tutte. L'ancora scarsità d'utilizzo a livello industriale di approcci agent-oriented certamente non ha aiutato in questo senso, dal momento che non ha incoraggiato a stabilire una qualche forma di standard nella definizione di metodologie ad agenti, così come,

viceversa, proprio questa mancanza di uno standard scoraggia il mondo industriale ad adottare tali metodologie, e quindi ancora il paradigma ad agenti in generale.
Lo scenario attuale pone dunque l'ingegneria del software orientata agli agenti in una duplice sfida: da una parte c'è ancora la necessità di testare e verificare le metodologie fino ad ora proposte, soprattutto quelle nuove, anche se spesso perfino quelle ormai ben consolidate necessiterebbero di una maggiore attenzione, dall'altra invece c'è un crescente bisogno di un qualche metodo di "valutazione" delle metodologie, che guidi i progettisti di sistemi nella scelta di quella più idonea ai loro progetti.
Quel che si potrebbe essere tentati di fare è di cercare una metodologia principe fra le quelle disponibili, cioè che possa essere, in ogni possibile caso, migliore delle altre. In realtà già per quanto riguarda l'analisi delle metodologie per il processo di sviluppo in generale, la sofware engeneering ha già messo in luce che una tale metodologia non può esistere: i requisiti sul progetto e i vincoli sulle risorse di produzione in genere limitano fortemente la scelta fra le metodologie esistenti. Il compito essenziale del metodo di valutazione dovrebbe essere quello di mettere in luce i maggiori pregi e difetti di ciascuna metodologia, rispetto alle altre, e i concetti riguardanti agenti e sistemi multi-agente sul quale essa si concentra maggiormente. In base ai risultati di questa valutazione l'ingegnere/progettista sceglierà quella che più si adatta allo specifico progetto.

2. Metodologie Orientate agli agenti

2.1. La panoramica attuale delle metodologie orientate agli agenti

Da quando è stato definito per la prima volta il concetto di agente nel campo della software engeneering fino ad oggi, sono state definite decine metodologie per lo sviluppo di software fondate su tale paradigma. Quando si sviluppa una nuova metodologia in genere si parte da una base di idee appartenenti ad altre metodologie esistenti, i concetti delle quali vengono in un qualche modo estesi. Come per le metodologie in generale dunque, così anche per quelle orientate agli agenti esiste una sorta di genealogia che ne riassume la storia; in [2] si è provato a dare una rappresentazione grafica di tale albero genealogico, riportata in figura 1.

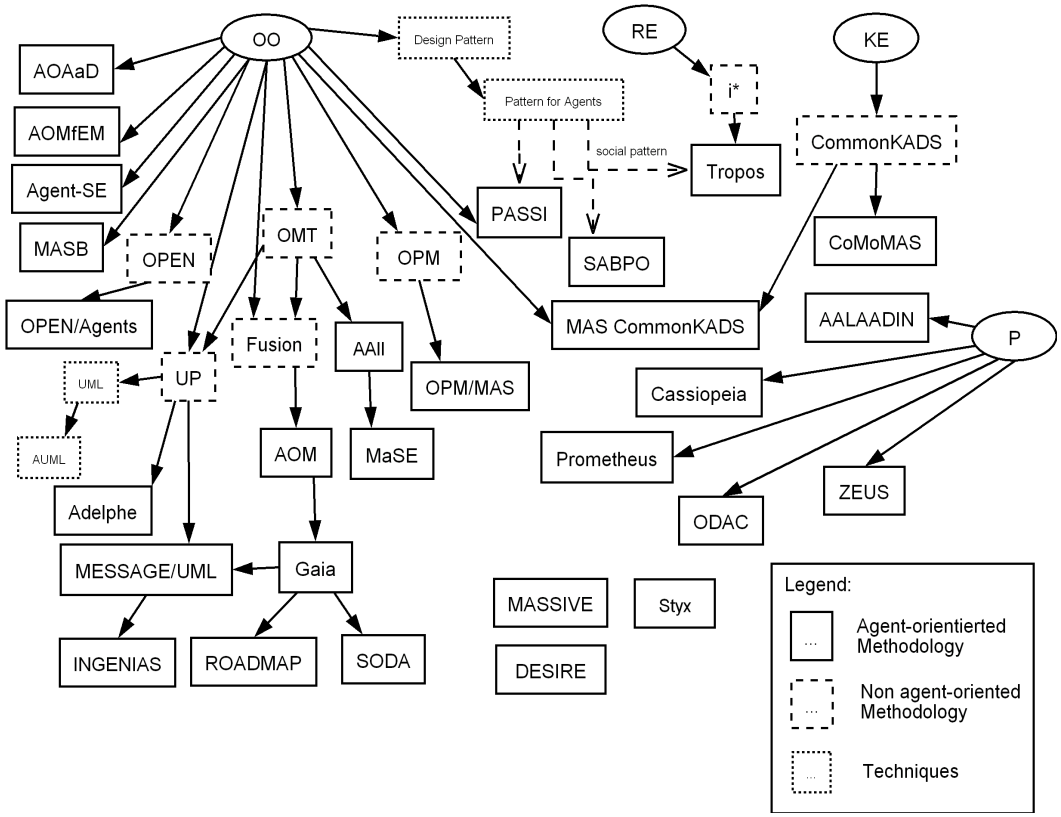


Illustrazione 1: genealogia delle metodologie agent-oriented

In ovale sono rappresentate le categorie di metodologie base dalle quali le prime metodologie ad agenti sono derivate e, come si può ben notare, la base di derivazione più comune è costituita dalle metodologie orientate agli

oggetti (OO); se consideriamo il concetto d'agente come, in un certo senso, un'evoluzione di quello d'oggetto, ciò sarebbe anche abbastanza prevedibile. Le altre categorie base, di minore ispirazione ma dalle quali sono comunque state derivate metodologie agent-oriented di rilievo, come Tropos o Prometheus, sono relative all'ingegneria dei requisiti (RE), ingegneria della conoscenza (KE) e un altro insieme di metodologie eterogeneo (P). Ovviamente ci si attende che tanto più due metodologie hanno "origini" diverse tanto più saranno anche differenti fra loro. Quindi per fornire una panoramica quanto più generale e allo stesso tempo completa occorrerebbe andare ad analizzare almeno alcune metodologia per ogni tipologia. Nella presente relazione comunque andremo ad analizzarne solo alcune.

2.2. Alcune metodologie orientate agli agenti

Le metodologie che si è deciso prendere come esempio, e che saranno oggetti anche nella parte relativa alla valutazione delle metodologie, sono fra le più conosciute e utilizzate nell'ambito dello sviluppo del software orientato agli agenti, quindi anche fra le più testate: MaSE, Prometheus, Tropos, Gaia. Inoltre ci si concentrerà anche su SODA, che rappresenta invece un esempio di metodologia relativamente più recente. Andiamo a fare dunque una veloce panoramica su queste metodologie.

Mase

La metodologia MaSE (Multiagent System Engeneering) è stata ideata a partire dal gruppo di metodologie AAII, a loro volta sviluppate a partire da metodologie basate su concetti object-orieted. MaSE descrive dettagliatamente il processo di sviluppo del software, il cui ciclo di vita va dall'analisi delle specifiche del sistema fino alla sua implementazione; questo si compone nel complesso in due fasi principali, divise a loro volta in fasi minori. La fase di *analisi* si compone nella definizione dei goal, nell'applicazione dei casi d'uso e nel raffinamento dei ruoli, nel quale vengono prodotti il modello dei ruoli e il modello dei task concorrenti; la fase di *disegno* è caratterizzata invece dalla creazione delle classi agente, tramite la costruzione, a partire dal modello dei ruoli, del diagramma della classe, dalla costruzione dei mezzi di comunicazione, dall'assemblaggio delle classi agente, in cui si definisce l'architettura del sistema, e infine dal disegno del sistema complessivo, nel quale si specificano le locazioni dei singoli agenti. MaSE viene supportata da AgentTool, che nella sua ultima versione supporta tutte le sette fasi della metodologia.

Prometheus

Prometheus è una metodologia agent-oriented dettagliata, ma allo stesso tempo rivolta anche ad utenti non esperti, tanto che viene usata con successo anche da studenti non graduati. Si compone in tre fasi principali: *specifica del sistema*, *disegno architetturale* e *disegno dettagliato*. La specifica del sistema coinvolge le attività di determinazione dell'ambiente del sistema e di determinazione dei goal e delle funzionalità del sistema. L'ambiente è definito in termini di percezioni e azioni, ossia di informazioni che gli agenti ricevono dall'ambiente e di azioni che gli agenti possono effettuare sull'ambiente stesso. In effetti Prometheus è uno dei primi linguaggi a prendere esplicitamente in considerazione l'entità ambiente. Il disegno architetturale è composto dalle seguenti attività: definizione dei tipi di agente, disegno della struttura complessiva del sistema e definizione delle interazioni fra agenti. La definizione dei tipi viene effettuata specificando le funzionalità degli agenti e raggruppandoli di conseguenza, mentre il disegno della struttura complessiva viene mostrato tramite il diagramma di panoramica del sistema, che rappresenta il più importante fra gli artefatti di lavoro di Prometheus. Infine la fase di disegno dettagliato si concentra sulle definizioni di capacità, eventi interni, piani e strutture dati dettagliate, per ogni tipo di agente. Esistono due tool per operare con Prometheus, in particolare per quel che riguarda la fase di disegno, JACK Development Tool (JDE), inerente alla piattaforma JACK, e Prometheus Design Tool (PDT); nessuno dei due si occupa comunque della fase dei requisiti.

Tropos

La metodologia Tropos è (almeno fra quelle di una certa rilevanza) l'unica che deriva quasi direttamente (anche) dalle tecniche dell'ingegneria dei requisiti, e per questo, più delle altre metodologie, pone particolare attenzione proprio alla fase di *analisi dei requisiti*; infatti questa attività si compone in ben due fasi principali dell'intera metodologia. Il processo di sviluppo di Tropos consiste quindi nei *primi requisiti*, nei *tardi requisiti*, nel *disegno architetturale*, nel *disegno dettagliato* e nell'*implementazione*. Nella prima analisi dei requisiti vengono costruiti il diagramma degli attori, in cui vengono elencati gli attori e le loro interdipendenze, e il diagramma dei goal; in particolare Tropos distingue fra due tipi di goal, hard-goal e soft-goal, in base al loro grado di soddisfacibilità. Nella seconda fase di analisi dei requisiti i precedenti diagrammi vengono ulteriormente estesi, in particolare raffinando le interdipendenze fra attori e scomponendo i goal in sotto-goal, allo scopo di adattare il sistema alle caratteristiche dell'ambiente. In fase di disegno architetturale viene effettuata la scelta dello stile architetturale e quindi adattata ad essa i modelli di attori e goal fin qui creati, mentre in fase di disegno dettagliato vengono definite le

esatte specifiche degli agenti e delle loro interazioni a micro-livello. Infine Tropos si occupa anche dell'implementazione ossia del mapping fra strutture astratte e parti in linguaggio reali. La metodologia è stata concepita fin dall'inizio per funzionare su una piattaforma specifica, ossia JACK, una piattaforma studiata sulla base del modello BDI, e che costituisce dunque la piattaforma ideale per la sua implementazione.

Gaia

Nata come estensione della metodologia AOM (Agent-Oriented Methodology), che a sua volta ha radici nelle metodologie object-oriented, come per MaSE, Gaia pone particolare attenzione negli aspetti gestionali che riguardano il processo di sviluppo. Inizialmente concepita per aver a che fare solo con le fasi di requisiti (anche se solo in parte) e analisi del processo di sviluppo stesso, è stata in seguito estesa per trattare anche con la fase di disegno. Si articola dunque nelle fasi dei *requisiti*, di *analisi*, di *disegno architetturale* e *disegno dettagliato*. La fase di analisi dei requisiti è in vero piuttosto semplice e viene trattata solo in maniera implicita. La fase di analisi invece si articola nelle attività di definizione dell'organizzazione, definizione del modello dell'ambiente, definizione del modello dei ruoli preliminare, suddivisi fra permessi e responsabilità, definizione del modello delle interazioni preliminare e definizione delle regole di organizzazione. Come, ad esempio, Prometheus, anche Gaia è una delle prime metodologie a tenere conto dell'ambiente, anche se non in maniera approfondita, inoltre si può considerare come la prima in assoluto che va a considerare le società di agenti (qui chiamate “organizzazioni”) come entità di prima classe. Il disegno architetturale completa i modelli dei ruoli e delle interazioni, fornendo anche una rappresentazione grafica. Infine il disegno dettagliato definisce i modelli degli agenti, a partire da quello dei ruoli, e il modello dei servizi.

SODA

SODA è una metodologia relativamente recente, ideata e sviluppata a partire da Gaia e che quindi eredita alcune peculiarità da questa, come ad esempio la forte attenzione che pone nel modellare le società di agenti e l'ambiente; per quel che riguarda il concetto di ambiente anzi, SODA pone ancora maggiore attenzione, modellandolo come un'entità di prima classe e con caratteristiche simili a quelle di un agente. Quel che invece non viene trattata è l'implementazione interna dei singoli agenti: di fatti SODA può considerarsi una metodologia incompleta, dal momento che si occupa solo di problemi *inter-agenti* e non *intra-agenti*. SODA porta dunque a definire un sistema multi-agente in termini del comportamento osservabile di agenti e società che lo compongono, e dei ruoli degli agenti nelle singole società.
Il ciclo di sviluppo si articola essenzialmente in due fasi distinte: fase di

analisi e fase di *disegno*, divise a loro volte in fasi minori. Nel complesso si ha *analisi dei requisiti*, *analisi*, *disegno architetturale* e *disegno dettagliato*. In ciascuna delle due fasi principali vengono elaborati differenti modelli. Durante la fase di analisi vengono elaborati il modello dei ruoli, che esprime i singoli task associati ad un dato ruolo, il modello delle risorse, che esprime le funzionalità messe a disposizione delle risorse dell'ambiente, e il modello delle interazioni che indica appunto le interazioni che intercorrono fra ruoli e risorse (quali ruoli utilizzano quali risorse) e in che modo. Tali modelli sono dunque utilizzati in fase di disegno per la definizione dei modelli degli agenti, delle società e dell'ambiente.
SODA è stata concepita principalmente per lo sviluppo e il disegno di sistemi web-based, seguendo l'approccio orientato agli agenti, in particolare per aver a che fare con le problematiche relative a distribuzione, apertura, eterogeneità, situadness (la proprietà di essere situati) e mobilità, ed è sembrata fin da subito ben adatta per adempiere a tale scopo. Il problema di SODA è però che si tratta di una metodologia relativamente giovane e, nonostante sia già stata utilizzata per progetti inerenti ad applicazioni web, relativamente ancora carente di approfonditi casi di studio e di una certa formalizzazione, oltre che di dati relativi ad una sua valutazione. Scopo del presente lavoro è anche quello di capire come potrebbe essere effettuata una sua approfondita valutazione e dei risultati che da questa ci si potrebbe attendere sulla base delle conoscenze già in nostro possesso.

3. Testing delle metodologie orientate agli agenti

3.1. Circa il test di metodologie orientate ad agenti: contesto e motivazioni

Una volta ideate e sviluppate le metodologie hanno bisogno di essere testate, al fine di verificarne il funzionamento. Testare una metodologia significa applicarla a diversi esempi/casi di studio, in diversi scenari, e verificare i risultati ottenuti dal processo di sviluppo, non solo in termini del software obbiettivo ottenuto alla fine, ma anche in termini di comprensibilità del processo di sviluppo stesso, di tempo e di risorse impiegate. I risultati ottenuti servono dunque per un ulteriore messa a punto e perfezionamento; questa è un'attività che nel complesso può richiedere più tempo della stessa ideazione e stesura della metodologia.
Al fine di ottenere un testing significativo comunque, occorre applicare la metodologia a casi di studio opportuni, analizzando tutti i possibili scenari d'interesse, in modo da riuscire ad analizzare tutte le potenzialità della metodologia stessa. Metodologie orientate agli agenti richiederanno opportuni casi di studio, che tengano in considerazione delle loro peculiarità, per poter essere analizzate e testate efficacemente. Come si è ampiamente analizzato però tali metodologie coprono nel loro complesso una categoria assai ampia, e possono differire molto per concetti e tecniche, nel modo in cui esse vanno a definire il processo di sviluppo del software, quindi un esempio ideale per tutte queste sarebbe difficilmente individuabile, e occorrerebbe invece ricercarne ogni volta uno o più specifici.

3.2. Un metodo per la selezione di esempi significativi

In [1] Yu e Cysneiros hanno cercato di individuare delle linee guida per la selezione di esempi ad hoc, adatti sia per la comprensione che per l'analisi di metodologie di tipo agent-oriented. In base a quanto emerge dalla ricerca, un buon esempio si compone di un set di più scenari applicativi, coadiuvato da una serie di domande che devono essere poste alle singole metodologie. Esiste inoltre una serie di criteri utili per la selezione di un esempio efficace (nel contesto di metodologie orientati agli agenti).

Secondo tali criteri un buon esempio dovrebbe:

- *avere un ambito di competenze chiaro*
- *esibire chiaramente l'orientamento ad agenti*
- *riflettere gli aspetti del mondo reale*
- *essere rappresentativo di un determinato dominio applicativo*
- *permettere paragoni fra metodologie orientate ad agenti e metodologie non orientate ad agenti*
- *non essere condizionante nella scelta di un dato approccio*
- *avere una preminenza per problemi e criteri generici dell'ingegneria del software*
- *incentrarsi anche su problematiche di tipo non tecnico*

Nell'articolo viene proposto come esempio il progetto per lo sviluppo di un software appartenente al dominio delle applicazioni riguardanti la cura della salute, corredato da un vasto set di possibili scenari, divisi in due categorie a seconda che siano applicabili prima o dopo il completamento del processo di sviluppo, cioè rispettivamente in fase di sviluppo o di mantenimento / aggiornamento del software stesso. Dal set di possibili scenari si ricavano allora una serie di domande utili all'analisi dell'efficacia di ogni singola metodologia. Dal momento che per ogni scenario sono associate in genere più domande, e che il set di scenari è già di se piuttosto articolato, si tratta in vero di un elenco di domande abbastanza lungo ed esaustivo. E' possibile però classificare queste domande in base a determinati criteri di classificazione che coinvolgono caratteristiche proprie dell'orientamento ad agenti; tali criteri si dividono fra quelli che coinvolgono aspetti riguardanti la fase di sviluppo del software e quelli successivi a tale fase. Ne elenchiamo solo alcuni, d'altronde gli autori stessi affermano che tale set è in continuo aggiornamento sulla base dei diversi feed-back ricevuti: fra i criteri inerenti alla fase di sviluppo vi sono pro-*attività*, *autonomia di ragionamento*, *mobilità*, *concorrenza*, *numero di livelli d'astrazione*, *dominio d'analisi*, *evoluzione del database*, *rintracciabilità dei requisiti*, *supporto per i tools*, *integrazione con altre metodologie*, etc; fra quelli inerenti alle fasi successive il processo di sviluppo invece *flessibilità a evoluzioni e riutilizzi*, *gestibilità di progetti*, *capacità di produrre versioni leggere per problemi semplici*, etc.

3.3. Casi di studio per il testing di metodologie orientate ad agenti

Le metodologie per lo sviluppo di software orientato ad agenti più famose, fra le quali anche quelle analizzate in precedenza, sono già state ampiamente utilizzate per diversi progetti di sviluppo di applicazioni

software, specialmente applicazioni web, quindi per queste si dispone già di un certo set di casi di studio che può essere analizzato.
In [9] viene proposto un esempio di applicazione della metodologia Prometheus, tramite anche l'utilizzo del Prometheus Design Tool, per lo sviluppo di un'applicazione per la gestione di conferenze. Il caso di studio è utile per valutare, oltre che la metodologia stessa, anche il tool su cui essa fa affidamento. In [10] invece viene proposto un caso di studio per Tropos, che propone lo studio della fase di analisi dei requisiti per lo sviluppo di un software per la gestione aziendale e di rifornimenti. In entrambi gli esempi gli autori danno una valutazione finale nel complesso assai positiva riguardo la validità riscontrata delle metodologie.
In [11] viene proposto un caso di studio per SODA, per lo sviluppo di un software per conferenze simile a quello appena proposto per il caso di studio di Prometheus; è interessante in questo caso fare un confronto fra i risultati dei test delle due metodologie. Nell'esempio in questione comunque non vengono riportati risultati o valutazioni finali.
Come già detto mentre metodologie sviluppate da più tempo, e quindi attualmente più diffuse, come Prometheus e Tropos, siano già state largamente utilizzate per diversi progetti, e quindi opportunamente testate, altre, relativamente più recenti, richiedono ancora di essere ulteriormente considerate e verificate, quindi rimane ancora diverso lavoro da fare nell'ambito del testing di metodologie agent-oriented, e SODA in particolare, proprio perché giovane, ricade fra quelle alle quali va ancora dedicata una certa attenzione. Questo ovviamente nella considerazione che anche per le metodologie ormai maggiormente consolidate è ancora richiesto un continuo lavoro di aggiornamento e perfezionamento.

4. Valutazione delle metodologie orientate agli agenti

4.1. Sulla valutazione di metodologie orientate agli agenti: contesto e motivazioni

Passiamo ora all'altro emergente problema riguardante le metodologie nel contesto dell'ingegneria del software orientata agli agenti. Anche dopo averle ampiamente testate, ed averne fatto una opportuna scrematura, tenendo solo quelle che si è deciso di considerare veramente valide, in questo scenario tecnologico abbiamo ancora un grande numero di possibili metodologie orientate agli agenti per la scelta di un processo di sviluppo software, e tale numerosità può senz'altro essere un problema per il progettista software, nel momento in cui questi deve affrontare tale scelta. Risulterebbe a questo punto di enorme utilità poter analizzare tutti i pro e i contro delle varie metodologie, in tutte le caratteristiche che riguardano il modello generale di una metodologia orientata agli agenti, in modo da poter valutare la "bontà" di ciascuna di esse e fornire al progettista un utile metro di valutazione che lo guidi nella sua scelta. Ovviamente questa è fortemente condizionata dallo specifico problema, dal momento che ciascuna metodologia può eccellere più in certi aspetti rispetto ad altri, aspetti che possono essere proprio quelli d'interesse. Un grande impegno è stato dunque impiegato negli ultimi tempi prima nella ricerca dei criteri che potevano essere alla base di tale valutazione, poi nel tentativo di costruire una qualche forma di metodo o framework in grado di eseguire tale operazione in maniera più o meno automatica, applicando tali criteri. Diverse soluzioni sono state proposte, dalla comunità dei ricercatori, anche se in generale piuttosto differenti fra loro, almeno nell'approccio al problema; sembra infatti che ancora manchi un accordo comune in merito a quali siano tali criteri, anche se alcune idee sono sembrate essere migliori delle altre, tanto da venire anche usate come base per ulteriori approfondimenti da parte di altri ricercatori. In [2] gli autori propongono un elenco di quelle che ritengono essere state le migliori soluzioni individuate. Riportiamo dunque in seguito i risultati di tali ricerche.

4.2. Criteri e framework per la valutazione di metodologie orientate ad agenti

In [3] Malley e De Loch analizzano il problema da una prospettiva più generale rispetto a quel che riguardava solamente l'ingegneria del software orientata agli agenti, cercando di determinare i criteri utili non solo per scelta di una metodologia agent-oriented fra le tante, ma anche per valutare, per un dato problema, qualora una metodologia agent-oriented fosse effettivamente preferibile ad una object-oriented. In linea di massima questi criteri vengono suddivisi in due categorie. La prima riguarda quelli relativi ai problemi di gestione (del processo di sviluppo) e sono:

- *costo di acquisizione della metodologia*
- *costo di acquisizione dei tools di supporto*
- *disponibilità di componenti riutilizzabili*
- *effetti sulle pratiche dell'organizzazione degli affari*
- *fedeltà agli standard*
- *tracciabilità dei cambiamenti*

La seconda invece è relativa ai requisiti di progetto:

- *integrazione con i sistemi ereditari*
- *distribuzione*
- *ambiente*
- *struttura del sistema dinamico*
- *interazione*
- *scalabilità*
- *agibilità e robustezza*

Per tentare di dare una ulteriore motivazione al loro lavoro, gli autori hanno anche cercato di validare i criteri da loro individuati tramite un sondaggio, rivolto a organi competenti quali società, università o compagnie informatiche industriali, nel quale si chiedeva una valutazione per ogni criterio. I risultati sono stati poi d'aiuto anche per la successiva creazione di un framework di valutazione. Tale framework valuta una metodologia tenendo in considerazione ciascun criterio in maniera diversa, ossia associando una priorità maggiore a quei criteri che avevano trovato un maggiore consenso da parte della comunità.

In [4] Cernuzzi e Rossi hanno individuato un set di criteri che hanno poi ordinato in una serie di categorie e sotto-categorie, in modo da definire così una sorta di *albero degli attributi*. L'albero è diviso in tre rami principali, ciascuno suddiviso a sua volta in altri sotto-rami.
Attributi interni, che riguardano il modo in cui viene modellata la struttura interna degli agenti:

- *autonomia*
- *reattività*
- *pro-attività*
- *nozioni mentali*

- *credenze (beliefs)*
- *goals/desideri*
- *intenzioni*

Attributi d'interazione:

- *abilità sociale*
  - *relazioni organizzazionali fra gli agenti*
  - *interazione con gli altri agenti*
  - *tipi di interazioni fra agenti*
  - *committenze*
  - *comunicazione con gli altri agenti*
  - *interfacce con gli altri agenti*
- *interazione con l'ambiente*
- *controllo multiplo*
- *interessi multipli*
- *interazione dei sotto-sistemi*

Infine abbiamo gli attributi relativi a tutti gli altri requisiti del processo:

- *modularità*
  - *decomposizione*
  - *dipendenza dei modelli*
- *astrazione*
  - *astrazione all'interno di ciascuna fase*
  - *esistenza di primitive di disegno e di meccanismi di astrazione*
- *vista di sistema*
- *supporto di comunicazione*
  - *modelli chiari e precisi*
  - *transizioni sistematiche*

Partendo da tale modello gli autori hanno definito anche un framework di valutazione, il cui funzionamento si articola in tre fasi, suddivise a loro volta in passi successivi, per un totale di sei passi. La particolarità di tale framework è il modo in cui applica ad una prima valutazione di tipo qualitativo, un'analisi di tipo quantitativa. Nella fase di *definizione* viene costruito l'albero degli attributi, seguendo il modello proposto ed eventualmente adattandolo ai requisiti specifici del progetto, e le metriche e delle scale di valori da utilizzare per la valutazione degli attributi stessi. Nella fase di *misurazione* vengono misurati gli attributi che corrispondono alle foglie nell'albero degli attributi, e vengono valutati secondo la metrica e la scala di valori più opportuna; successivamente vengono ricavati anche i valori di valutazione degli attributi radici secondo certe tecniche di derivazione. Infine nella fase di *analisi* vengono analizzati i risultati così ottenuti per stillare una valutazione finale della metodologia.

La soluzione proposta da Dam e Winikoff in [6] è frutto della ricerca da parte degli autori su altri metodi di confronto proposti, sia fra metodologie object-oriented che fra quelle agent-oriented. I criteri da loro individuati si possono raggruppare in quattro categorie, che coprono praticamente tutti gli aspetti essenziali di una metodologia orientata agli agenti: *concetti*, *linguaggio di modellazione*, *processo* e *pragmatici*. I concetti sono la base per la definizione di una metodologia, e riguardano fra l'altro la definizione stessa di agente e le sue caratteristiche:

- *autonomia*
- *attitudini mentali*
- *pro-attività*
- *reattività*
- *concorrenza*
- *lavoro di gruppo*
- *protocolli*
- *collocazione*
- *concetti chiari*
- *orientamento ad agenti*

Un linguaggio di modellazione descrive in che modo la metodologia è costruita in termini dei concetti appena descritti; i criteri ad esso relativi sono suddivisibili in criteri di usabilità e criteri tecnici:

- s*tatica o dinamica*
- *sintassi definita*
- *semantica definita*
- *notazione chiara*
- *facilità d'uso*
- *facilità d'apprendimento*
- *viste differenti*
- *adeguatezza ed espressività del linguaggio*
- *tracciabilità*
- *controllo di consistenza*
- *raffinamento*
- *modularità*
- *riuso*
- *modellazione gerarchica*

I criteri di processo descrivono l'intero ciclo del processo di sviluppo, analizzando se e in che misura ciascuna delle fasi tipiche viene presa in considerazione:

- *requisiti*
- *analisi*
- *disegno architetturale*

- *disegno dettagliato*
- *implementazione*
- *test e debugging*
- *deployment*
- *mantenimento*

Infine i pragmatici riguardano gli aspetti più propriamente pratici, come ad esempio quelli relativi ai tools associati ad una metodologia, e sono divisi fra quelli inerenti problemi tecnici e problemi di gestione:

- *qualità*
- *costo stimato*
- *decisioni di gestione*
- *applicazioni e applicazioni reali*
- *utilizzo al di fuori dei creatori*
- *dominio specifico*
- *scalabilità*
- *distribuzione*

Il framework di valutazione in questo caso consiste nella verifica di quanto ciascun criterio venga riscontrato in una metodologia. Viene dunque stillata una scala discreta di valori per verificare ad esempio se un dato criterio è fortemente riscontrato, mediamente riscontrato, poco riscontrato, per niente riscontrato, etc. Al fine di avere una valutazione il più completa e imparziale possibile gli autori suggeriscono di procedere all'analisi degli attributi di una metodologia attraverso una forma di sondaggio, rivolto sia agli sviluppatori che agli utilizzatori della metodologia stessa. Tale fase è di fatti una delle più delicate in un qualunque processo di valutazione di una metodologia, dal momento che esiste un rischio non indifferente di cadere in considerazioni prettamente soggettive o imparziali (specialmente quando chi valuta è anche lo sviluppatore della metodologia). Un'analisi delle caratteristiche tramite sondaggio permette una buona imparzialità e correttezza, anche se allo stesso tempo si rivela dispendiosa. I criteri, e la relativa classificazione, individuati in questa ricerca, sono sembrati subito molto validi agli occhi degli altri ricercatori, tanto da venir più volte utilizzati per successive ricerche per la valutazione di metodologie agent-oriented.

Shehory e Sturm ad esempio, utilizzarono tali risultati ai fini di ampliare la loro prima ricerca. Nel loro primo lavoro, descritto in [6], essi classificano i criteri di valutazione in due categorie, semplicemente suddividendoli fra quelli inerenti ai metodi di classificazione delle metodologie classiche dell'ingegneria del software, e quelli che trattano aspetti propri dei sistemi multi-agente. I criteri di valutazione dell'ingegneria del software sono:

- precisione

- accessibilità
- espressività
- modularità
- gestione della complessità
- eseguibilità (e testabilità)
- raffinabilità (e implementatività)
- analizzabilità
- apertura

Mentre criteri relativi alle caratteristiche di sistemi basati su agenti sono:

- *autonomia*
- *complessità*
- *adattabilità*
- *concorrenza*
- *distribuzione*
- *ricchezza della comunicazione*

In seguito, in [7], gli autori ripresero questi risultati, combinandoli con quelli di Dam e Winikoff in [5], arrivando alla definizione di nuovi criteri e di un nuovo framework di valutazione. In vero si tratta alla fine di una soluzione molto simile a quella proposta in [5], anche per quanto riguarda il significato di ogni criterio.

Abbiamo criteri relativi a concetti e proprietà, suddivisi a loro volta fra concetti base:

- *autonomia*
- *reattività*
- *pro-attività*
- *socialità*

e blocchi base di costruzione, che costituiscono concetti tipici di un sistema multi-agente

- a*genti*
- *credenze (belief)*
- *desideri*
- *intenzioni*
- *messaggi*
- *norme*
- *organizzazioni*
- *protocolli*
- *ruoli*
- *servizi*
- *società*
- *task*

criteri delle notazioni e tecniche di modellazione:

- *accessibilità*
- *analizzabilità*
- *gestione della complessità*
- *eseguibilità*
- *espressività*
- *modularità*
- *precisione*

criteri relativi al processo di sviluppo

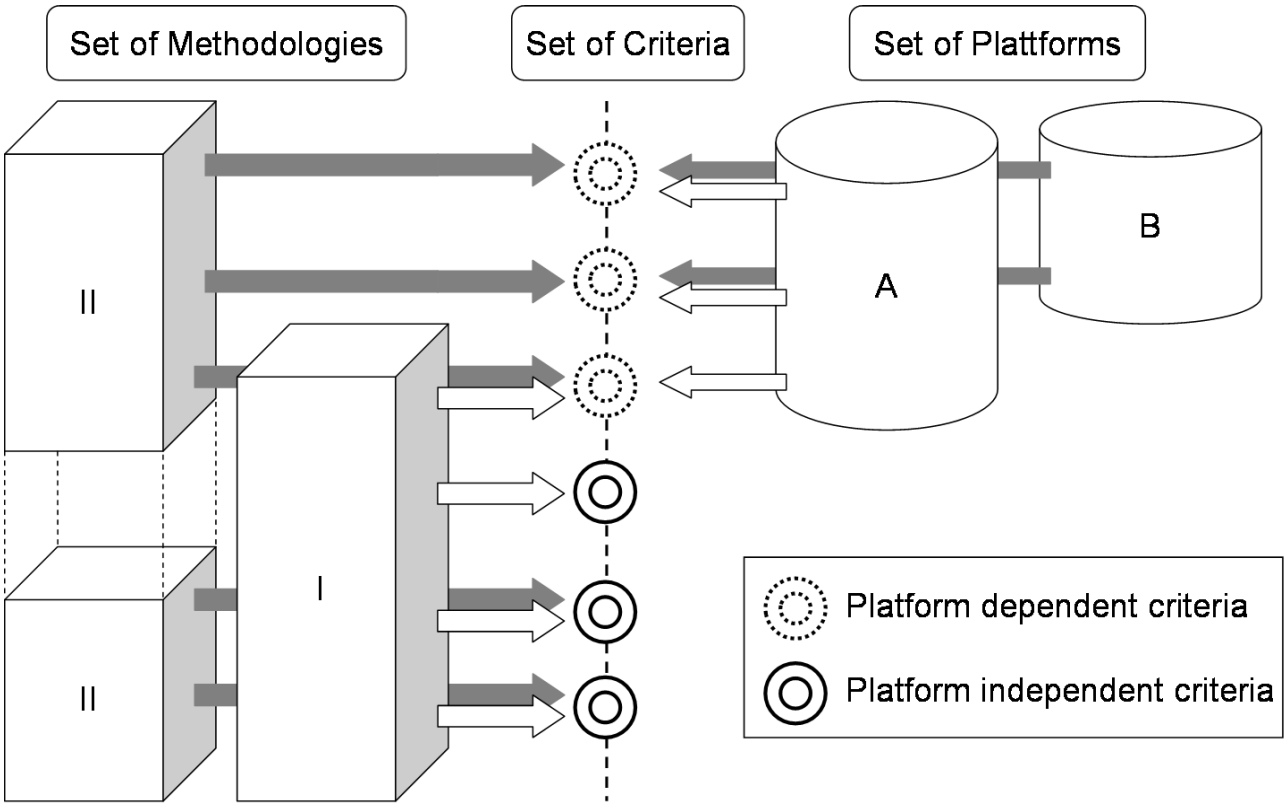
- *contesto del ciclo di sviluppo*
- *copertura del ciclo di vita dello sviluppo (requisiti, analisi, disegno, implementazione e testing)*

infine criteri relativi a pragmatici:

- *risorse*
- *esperienza richiesta*
- *adattabilità del linguaggio (paradigmi ed architettura)*
- *applicabilità del dominio*
- *scalabilità*

Il framework prevede allora la definizione di una scala metrica discreta di valori, in un range adeguato, in modo da assegnare una graduatoria che indichi quanto ciascun criterio viene considerato. Tale framework di valutazione si è rilevato in effetti molto buono nell'esperienza e adatto ad evidenziare vantaggi e svantaggi di ciascuna metodologia, soprattutto grazie al fatto di basarsi su ricerche precedenti a loro volta assai valide.

In [2] Sudeikat, Braubach, Pokahr e Lamersdorf, pongono attenzione su una nuova questione non trattata nei precedenti lavori: secondo quanto sostenuto da loro, ai fini di una corretta valutazione, una metodologia non può essere analizzata senza considerare anche la piattaforma utilizzata per applicarla, dal momento che, in generale, vi sono differenze considerevoli fra le diverse implementazioni che le diverse piattaforme offrono. Secondo tali considerazioni vi sono dunque due tipi di criteri per la valutazione: *criteri dipendenti dalla specifica piattaforma* e *criteri indipendenti*. Prima di ricavare i primi è ovviamente necessario, in qualunque caso di studio possibile, andare ad analizzare le caratteristiche proprie della piattaforma (o delle piattaforme) scelta (scelte). I concetti di questo metodo vengono evidenziati in figura.



Sono possibili a questo punto tre diversi tipi di casi di studio: condurre una valutazione su un certo numero di metodologie implementate tutte su una sola piattaforma, valutare una sola metodologia prendendo in considerazione varie piattaforme sulle quale poterla implementare, oppure condurre una valutazione con una relazione di tipo *n-m* fra metodologie e piattaforme (*n* metodologie valutate su *m* piattaforme). Trovare i criteri di valutazione in questo contesto vuol dire anche trovare quali requisiti specifici di una metodologia vengono soddisfatti da una data piattaforma, e viceversa. I criteri di valutazione sono classificati nello stesso modo indicato da Dam e Winikoff in [5], con

- *concetti*: architettura interna, architettura sociale, comunicazione, autonomia, pro-attività e distribuzione.
- *notazioni*: utilizzabilità, espressività, raffinamento, dipendenza dei modelli, tracciabilità, definizione chiara e modularità.
- *processo*: copertura del flusso di lavoro (requisiti, analisi, disegno, etc.), gestione, complessità e proprietà di processo.
- *pragmatici*: supporto dei tools, connettività, documentazione e utilizzo in progetti.

I criteri relativi a notazioni e processo sono generalmente di tipo indipendenti dalla piattaforma, mentre per quanto riguarda concetti e (almeno in parte) pragmatici si tratta in genere di concetti dipendenti dalla specifica piattaforma. Il framework qui proposto prevede, dopo aver

definito la/le piattaforma/e d'interesse, e averne individuato le caratteristiche essenziali, che prima si mappino i concetti astratti relativi ai criteri dipendenti dalle piattaforme in attributi concreti, poi si esegua la valutazione tenendo anche conto dei concetti indipendenti dalla piattaforma. Inoltre viene proposto di effettuare la valutazione basata sui criteri relativi a concetti e processo, da una parte, e quella basata sui criteri relativi a modellazione a pragmatici, dall'altra, in due momenti differenti. La valutazione in base ai criteri relativi a concetti e progetto può essere fatta prima dell'applicazione di un esempio/caso di studio specifico, analizzando anche quanto gli aspetti di una piattaforma corrispondano a quelli della metodologia, e viceversa. L'analisi dei criteri relativi a notazioni e pragmatici invece si presenta assai più difficoltosa, anche perché spesso condizionata dai specifici tools CASE, e si rende più opportuno dunque effettuarla mediante uno specifico caso di studio o attraverso una forma di sondaggio.

4.3. Esempi di applicazione di framework di valutazione di metodologie agent-oriented

Consideriamo ora alcuni esempi di applicazione dei framework di valutazione precedentemente illustrati.

In [5] Dam e Winikoff propongono un esempio, per il framework da loro ideato, che dove viene effettuata una valutazione e un confronto fra le metodologie MaSE, Prometheus e Tropos. Basandosi sul loro modo di operare, già descritto in precedenza, le specifiche d'interesse relative a ciascuna di queste metodologie sono state individuate per mezzo di un questionario di 60 domande consegnato sia agli sviluppatori che a degli specifici utilizzatori, in questo caso studenti impegnati in un medesimo progetto. I risultati sono riassunti in tabella:

	MaSE	Prometheus	Tropos
Concetti			
Autonomia	H/M/DK	H/NA/H	H/M/M
Attitudini mentali	L/M/H	H/M/H	H
pro-attività	H/M/H	H/M/DK	H
reattività	M	H/M/DK	H/L/DK
concorrenza	H/M/H	M/L/DK	H/M/H
lavoro di squadra	H/M/H	N/L/NA	H/H/M
protocolli	H	M/H/M	NA/M/M

situadness	M/L/H	H	H
Concetti in chiaro	SA/A/A	A/A/DA	SA/A/N
Concetti overstimati	A/N/SA	N	SDA/N/DA
Orientamento ad agenti	SA/A/A	SA/A/A SA	SA/A/SA
Linguaggio di modellazione			
staticità/dinamicità	SA/A/A	SA/A/A	N/A/A
sintassi definita	A/A/SA	SA/A/A	SA/N/A
semantica definita	A/SA/SA	A	SA/A/A
notazione chiara	A	SA/A/A	SA/A/N
facilità d'uso	SA/A/A	A/N/A	SA/A/N
facilità d'apprendimento	N/N/A	SA/NA/SA	SA/N/A
differenza nelle viste	N/N/A	A/A/SA	SA/A/N
linguaggio adeguato ed espressivo	SA/N/N	A	SA/A/N
tracciabilità	A/SA/SA	A	A/N/A
controllo di consistenza	SA/A/SA	SA/A/A	_/A/DA
raffinamento	SA/A/A	SA	SA/A/DA
modularità	SA/A/A	SA/SA/A	SA/A/N
riutilizzo	N/SA/A	N/A/N	_/A/DA
modellazione gerarchica	N/A/A	SA/A/A	SA/A/DA
Processo			
requisiti	SPEH	SPEH	SPE
disegno architetturale	SPEH	SPEH	SPE
disegno dettagliato	SPEH	SPEH	SPE
implementazione	SEH/SPE/S	SPEH/S/n	SE/SPE/SPEH
testing e debugging	SPE/n/n	SPEH/S/n	n
deployment	SE/SPE/SPEH	n	n
mantenimento	n/SPE/n	n	n
Pragmatici			
qualità	N/DA/A	A/N/N	DA/A/_
stima del costo	_/DA/SA	DA/DA/N	DA/N/_
decisioni di gestione	_/DA/SA	SDA/N/_	SA/A/_

apps	21+	6-20	1-5
apps reali	No	No	no
utilizzo al di fuori dei creatori	Yes	yes	yes/no/no
specificità nel dominio	No	no	yes/no/no
scalabile	_/N/N	N/A/N	N/N/_
distribuito	_/SA/SA	SA/A/N	N/A/_

Legenda:

- L: basso
- M: medio
- H: alto
- DK: ignoto
- SDA: fortemente in disaccordo
- DA: in disaccordo
- NA: non applicabile
- N: neutrale
- A: applicabile
- A: in accordo
- SA: fortemente in accordo
- _: nessuna risposta
- S: fase menzionata
- P: processo dato
- E: esempio dato
- H: euristica data
- n: niente

Per ogni casella si hanno tre valori: i primi due corrispondono alle risposte date dagli sviluppatori, il terzo dall'utilizzatore, mentre se solo una è presente vuol dire che tutte e tre coincidono.

Andando ad analizzare i risultati ottenuti, si nota generalmente un certo livello di accordo riguardo agli attributi di ogni metodologia, anche se alcune forti discrepanze fra giudizio di sviluppatori e utilizzatori sono comunque presenti. Facendo una considerazione a livello generale si nota facilmente che fra le tre metodologie non ve ne è una prettamente migliore o peggiore delle altre, dal momento che queste hanno punti di forza o debolezze reciproche in diverse caratteristiche. Ad esempio a livello di concetti Prometheus e Tropos sembrano offrire una maggiore attenzione ai concetti di attitudini mentali rispetto a MaSE, mentre invece quest'ultima si dimostra migliore per i protocolli. Per determinate categorie di attributi però qualche metodologia si dimostra più performante delle altre due, ad esempio a livello di processo MaSE offre un buon supporto anche per il deployment e per il mantenimento. Stillando un giudizio finale dunque

tutte e tre le metodologie possono rivelarsi equamente valide per un progetto di sviluppo software, pur nella consapevolezza che qualcuna può eccellere sulle altre a seconda delle specifiche del particolare progetto di sviluppo.

Riportiamo ora l'esempio che gli autori del framework di valutazione proposto in [2] hanno utilizzato per avvalorare la loro teoria. Come già illustrato, tale framework tiene in considerazione sia dei criteri indipendenti dalla piattaforma specifica, sia di quelli dipendenti, ed analizza l'efficacia, nel caso generale, di *n* metodologie su *m* piattaforme. Nel caso in questione si considerano tre metodologie implementate su un'unica piattaforma, ossia le metodologie MaSE, Prometheus e Tropos, e la piattaforma Jadex. Jadex è un add-on della piattaforma ad agenti JADE, che la estende con efficaci e potenti meccanismi BDI. La prima operazione del framework prevede l'individuazione dei criteri dipendenti dalla piattaforma, ossia i requisiti della piattaforma, che necessitano di essere soddisfatti da ciascuna metodologia, e viceversa. In questo caso, lavorando su un'unica piattaforma, non si tratta di un problema troppo complesso. Tenendo in considerazione le caratteristiche di Jadex, i criteri individuati sono: per la parte dell'architettura interna *la struttura BDI* (*credenze*, *desideri*, *intenzioni*), *capacità ed eventi*, per l'architettura sociale i *ruoli* e per la parte di comunicazione i *protocolli* e i *messaggi*. La seconda fase del framework consiste nell'analisi dei criteri relativi a concetti e processo, i quali possono essere esaminati tramite semplice visualizzazione e senza avvalersi dell'esempio di uno specifico caso di studio. Nel complesso si tratta sia di criteri dipendenti dalla piattaforma che indipendenti (quelli relativi a concetti sono quasi tutti dipendenti, quelli relativi a processo esclusivamente indipendenti). Per quelli dipendenti dalla piattaforma, oltre ad analizzare quanto questi vengano considerati sia da metodologia che da piattaforma, va anche analizzato quanto valga il grado di corrispondenza, ossia quanto i concetti astratti espressi dall'una siano implementati in maniera coerente dall'altra. La tabella seguente mostra i risultati ottenuti; da notare che per criteri dipendenti da piattaforma accanto al valore corrispondente per una metodologia è indicato il grado di corrispondenza con Jadex:

	MaSE	Tropos	Prometheus	Jadex
Concetti				
Architettura interna				
goals	+ | /	+ | ~	+ | /	+
piani	+ | ~	+ | ~	+ | ~	+
credenze	+ | ~	+ | ~	+ | ~	+

capacità	- | /	+ | ~	+ | =	+
eventi	+ | ~	+ | ~	+ | ~	+
Architettura sociale				
ruoli	+ | /	+ | /	- | =	-
Comunicazione				
protocolli	+ | /	+ | /	+ | /	-
messaggi	+ | ~	+ | ~	+ | ~	+
autonomia	++	++	++	
pro-attività	+	++	++	
distribuzione	+ | =	- | /	- | /	+
Processo				
Copertura del flusso di lavoro	3 su 5	4 su 5	4 su 5	
gestione	n.a.	n.a.	n.a.	
complessità	++	++	++	
Proprietà del processo	Interattive e top-down	Interattive e top-down	Interattive e top-down	

Leggenda:
--: povero
-: non bene
+: bene
++: molto bene
n.a.: non disponibile
/: nessuna corrispondenza
~: corrispondenza approssimata
=: buona corrispondenza

L'ultima fase del framework consiste nell'analisi dei criteri relativi a notazioni (linguaggio di notazione) e pragmatici. Per fare ciò, in questo esempio, ci si avvale di un particolare caso di studio, che fra l'altro è compreso fra quelli del tutorial di Jadex. Si tratta comunque di una fase difficile e i risultati vengono brevemente riassunti, senza essere schematizzati. E' comunque abbastanza analoga all'analisi di linguaggio di modellazione e pragmatici del metodo di Dam e Winikoff quindi evitiamo di inoltrarci troppo. Limitiamoci soltanto a notare relativamente ai criteri pragmatici che, a livello di tools utilizzati, nessuna delle tre metodologie presente una buona connettività con la piattaforma Jadex.
Quel che si evince in questo caso, dalla valutazione finale, è che, nonostante tutte e tre le metodologie si dimostrano in grado di poter supportare lo sviluppo di applicazioni in Jadex, una fra queste andrebbe

preferita, ossia Prometheus, dal momento che è quella che presenta un maggior numero di aspetti a favore, fra i quali ad esempio una descrizione dettagliata dei singoli componenti e il tool, sul quale essa fa affidamento, liberamente disponibile. MaSE sembra invece essere la meno adatta, soprattutto per il supporto povero ai concetti BDI.

Per quanto riguarda SODA tentativi di valutazione e di confronto con altre metodologie sono ancora scarsamente riscontrabili. Nonostante questa sia già stata utilizzata per diversi progetti di sviluppo software e, almeno in maniera discreta, testata nel suo funzionamento, tali progetti in genere sono stati effettuati al di fuori dal contesto di una valutazione della metodologia stessa. In [12] viene proposta una semplice ed intuitiva valutazione di SODA ed un confronto con altre metodologie, fra le quali Gaia e Tropos. Vengono considerati solamente quattro semplici criteri di valutazione: *l'analisi dei requisiti*, *la modellazione della struttura sociale*, *la modellazione dell'ambiente* e *la gestione della complessità*.

	Analisi dei requisiti	**Modellazione struttura sociale**	**Modellazione ambiente**	**Gestione della complessità**
Gaia	Buona	Buona	Quasi assente	Assente
Tropos	Buona	Da migliorare	Assente	Assente
SODA	Buona	Buona	Buona	Buona

In questo esempio SODA pare rivelarsi generalmente migliore rispetto alle altre, in particolare per quanto riguarda la modellazione dell'ambiente e la gestione della complessità, praticamente assenti per le altre metodologie.
La sensazione nel caso appena mostrato tuttavia è più che altro quella che si tratti di un framework ad hoc per mettere in rilievo i vantaggi di SODA, essendo molto incentrato sugli aspetti in cui questa si mostra maggiormente promettente. Sarebbe interessante avere a disposizione una valutazione di SODA effettuata con i framework precedentemente introdotti, però riguardo a questi non sono ancora stati riportati casi concreti di utilizzo con la metodologia in questione, quindi considerazioni finali che siano reali ed oggettive non sono disponibili. Avendo a mente almeno la classificazione individuata da Dam e Winikoff tuttavia, si potrebbe comunque tentare di applicarne i criteri per cercare di effettuare una valutazione a priori, tenendo pur sempre presente che, per una valutazione tecnica veramente valida, questa sarebbe da effettuare nel contesto di un caso di studio reale. Le caratteristiche della metodologia che riguardano concetti e processo si rivelano assai intuitive da valutare: SODA si concentra fortemente sui concetti di *autonomia*, *pro-attività* e *reattività* (effettua una divisione fra le astrazioni relative ad attività attive e passive), così come dovrebbero avere una certa rilevanza concetti come la

cooperazione fra entità e *situadness*, dal momento che società ed ambiente sono trattate come entità di prima classe. SODA non sembra trattare gli stati mentali (il modello BDI) e sicuramente non a livello di singolo agente dal momento che è una metodologia concentrata solo sugli aspetti *inter-agenti*. Fortemente seguito è invece il meta-modello ad Agenti ed Artefatti. Per quanto riguarda il processo invece molto facilmente si può notare che le fasi seguite sono *requisiti*, *analisi* e *disegno*, quest'ultimo in una maniera particolare dato che viene diviso in due sotto-fasi. Sul linguaggio di modellazione e pragmatici invece una valutazione a priori si rende, come per qualunque altra metodologia, molto difficile e sarebbe invece meglio effettuarla nel contesto id un esempio concreto. Limitiamoci solo a qualche aspetto facilmente valutabile: ad esempio a livello di linguaggio di modellazione viene posta una particolare cura alla *gestione della complessità* e alla *modellazione gerarchica*, mentre per i pragmatici vi è ancora una evidente scarsità di tools e strumenti di modellazione efficaci.

5. Conclusioni

Nel corso di questa relazione abbiamo messo in luce quante diverse metodologie per lo sviluppo di software orientato agli agenti siano state ideate fino ad oggi, e in generale quanto queste possano differire fra loro, e questo anche per mettere maggiormente in luce quanto si rendano necessari oggi, nell'ambito dell'ingegneria del software orientata agli agenti, dei metodi di valutazione per le metodologie stesse, come primo mezzo di guida per effettuare la scelta. Quel che però è emerso è che, così come non esiste un modello standard per queste metodologie, non vi è un parere unanime sui criteri in base ai quali una metodologia deve essere valutata. Diversi tipi di criteri sono stati identificati e di conseguenza diversi framework di valutazione possibili sono stati concepiti, con la conseguenza che sembrerebbe essere tornati al punto di partenza, essendo ancora una volta di fronte ad una scelta. In realtà una certa forma di convergenza fra i lavori di diversi autori nella buona parte dei casi può comunque essere notata. In particolare quanto svolto da Dam e Winikoff in [5] viene ripreso più volte da altri autori e il loro modello di framework e di classificazione dei criteri può dunque venir considerato un buon passo verso una forma di standard.
Al di là di questa varietà comunque, e al di là del fatto che costanti miglioramenti continuano ad essere richiesti, i framework di valutazione individuati fino ad ora si sono rilevati, una volta messi alla prova, in linea generale assai validi e utili al fine di saggiare la "bontà" di una metodologia. Oltre a rappresentare, allo stato attuale delle cose, uno strumento estremamente prezioso per il progettista nella scelta di una metodologia, essi rappresentano anche un primo passo di miglioramento per quel che riguarda la situazione generale delle metodologie nell'ambito dell'ingegneria del software orientata agli agenti. Come già detto il problema sostanziale che ancora limita fortemente l'adozione di queste metodologie (e del paradigma ad agenti in generale) in campo industriale è la mancanza di modelli standard. L'idea sarebbe dunque quella di poter condurre una valutazione e un confronto, utilizzando tali framework, fra tutte queste metodologie, in modo da poterne effettuare una ulteriore scrematura ed arrivare ad eleggere quelle "migliori", adatte a rappresentare tali standard. Questo ovviamente sempre nella considerazione che una metodologia migliore delle altre in assoluto non può esserci e che le specifiche ed i vincoli imposti sul progetto costituiranno sempre un fattore discriminante.
Inoltre sarebbe sicuramente interessante applicare almeno qualcuno di questi framework di valutazione, già ampiamente utilizzati per

metodologie diffuse come Prometheus, Tropos, Gaia, etc., anche ad altre assai più recenti e fin'ora meno utilizzate, specialmente per i loro stessi sviluppatori i quali invece tendono generalmente a servirsi di altri framework e criteri da loro stessi definiti, quindi assai più soggettivi e imparziali. In particolare a noi interesserebbe molto, come già precedentemente affermato, una valutazione accurata ed oggettiva della metodologia SODA. D'altra parte ovviamente, specialmente per quanto riguarda queste stesse metodologie ancora poco utilizzate e spesso non perfettamente testate, parallelamente a questi lavori di valutazione e confronto, un continuo impegno per il loro miglioramento e il loro perfezionamento deve per forza di cose continuare ad essere sempre richiesto.

Bibliografia

[1] Eric Yu and Luiz Marcio Cysneiros. *Agent-Oriented Methodologies – Towards A Challenge Exemplar*

[2] Jan Sudeikat, Lars Braubach, Alexander Pokahr, and Winfried Lamersdorf. *Evaluation of Agent–Oriented Software Methodologies – Examination of the Gap Between Modeling and Platform.*

[3] Scott A. O'Malley and Scott A. DeLoach. *Determining When to Use an Agent-Oriented Software, Engineering Paradigm*

[4] Luca Cernuzzi and Gustavo Rossi. *On the Evaluation of Agent Oriented Methodologies*

[5] Khanh Hoa Dam and Michael Winikoff. *Comparing AgentOriented Methodologies*

[6] Onn Shehory and Arnon Sturm. *Evaluation of Modeling Techniques for Agent-Based Systems*

[7] Onn Shehory and Arnon Sturm. *A Framework for Evaluating Agent-Oriented Methodologies.*

[8] Andrea Omicini. SODA: *Societies and Infrastructures in the Analysis and Design of Agent-based Systems*

[9] Lin Padgham, John Thangarajah, and Michael Winikoff. *The Prometheus Design Tool - A Conference Management System Case Study*

[10] Maddalena Garzetti*, Paolo Giorgini*, John Mylopoulos., and Fabrizio Sannicolo. *Applying Tropos Methodology to a real case study: Complexity and Criticality Analysis*

[11] Andrea Omicini & Ambra Molesini. *Conference Management System: a Case Study in SODA*

[12] Ambra Molesini, Enrico Denti, Andrea Omicini. *Metodologie per l'ingegneria del software: approccio ad agenti*