

# Agent# : un linguaggio di programmazione per lo sviluppo di agenti su piattaforma .NET

C. Vecchiola, M. Coccoli, e A. Boccalatte

**Abstract**—Questo articolo descrive un linguaggio di programmazione orientato allo sviluppo di applicazioni ad agenti, denominato Agent#. Esso è stato progettato per offrire allo sviluppatore di applicazioni ad agenti un'interfaccia di programmazione più efficace per la progettazione e la realizzazione di agenti software. Agent# permette lo sviluppo di agenti in un determinato contesto esecutivo, la piattaforma AgentService. Interamente basato sulla tecnologia .NET sfrutta l'interoperabilità tra i diversi linguaggi per offrire un ambiente di programmazione professionale ed efficiente. In questo articolo verrà discusso come Agent# esponga gli elementi chiave del modello di programmazione ad agenti adottato dalla piattaforma AgentService, evitando che lo sviluppatore si occupi de dettagli tecnici ed implementativi di tale modello.

**Keywords**—Programmazione ad agenti, lingaggi di programmazione, componenti software.

## I. INTRODUZIONE

La tecnologia ad agenti è un argomento di ricerca ormai largamente diffuso. L'utilizzo di questo approccio è particolarmente adatto alla modellazione di sistemi complessi ed altamente dinamici in cui le interazioni possono non essere stabilite a priori. La realizzazione di modelli software in grado di rappresentare le realtà precedentemente descritte, trova la sua espressione più efficace nello sviluppo di un sistema multi-agente [1], [2]: una comunità di entità software autonome (gli agenti) che interagiscono allo scopo di perseguire i propri obiettivi.

Sebbene vi siano molte metodologie e strumenti per la realizzazione di agenti e di sistemi multi-agente, esse si possano inquadrare in due approcci di riferimento: vi sono modelli che impiegano la definizione di un agente e di un sistema multi-agente utilizzando tecniche di intelligenza artificiale e modelli che propongono un approccio più programmatico e sono maggiormente orientati allo sviluppo dei sistemi multi-agente proponendo framework per la programmazione di MAS. Un approccio complementare a

quelli precedentemente presentati è rappresentato dall'adozione di un linguaggio di programmazione orientato agli agenti. Tale approccio è di fatto complementare ai precedenti in quanto molto spesso il linguaggio di programmazione orientato agli agenti si appoggia ad un framework ad agenti, che costituisce il target del codice sviluppato con il linguaggio. E' quindi l'accoppiamento del linguaggio di programmazione e del framework che permettono la realizzazione e la messa in opera di un'applicazione orientata agli agenti.

Il lavoro discusso in questo articolo segue esattamente questo approccio: Agent# è infatti un linguaggio di programmazione orientato agli agenti ovvero, specificamente pensato per la realizzazione di agenti e non di software general purpose. Agent# si appoggia alla piattaforma ad agenti AgentService [7] ed i componenti con esso realizzati hanno come target tale piattaforma. Scopo del linguaggio Agent# è perciò astrarre il modello di agente esposto in AgentService attraverso opportune semantiche e sintassi. Agent# è stato pensato per essere un linguaggio di programmazione efficace e quindi uno dei criteri di progetto è stata l'integrazione con i componenti già esistenti per la programmazione delle applicazioni. Come base per lo sviluppo del linguaggio è stato scelto il framework .NET [4] che offre un supporto sia in termini di interoperabilità tra i differenti linguaggi di programmazione presenti nel framework sia in termini di risorse e componenti già presenti e facilmente integrabili nello sviluppo delle proprie applicazioni.

Con Agent# si vuole perseguire un duplice obiettivo: offrire un'interfaccia di programmazione maggiormente orientata agli agenti; automatizzare tutti i pattern di programmazione richiesti dalla piattaforma AgentService in modo trasparente all'utente del linguaggio. In questo modo gli sforzi dello sviluppatore sono maggiormente concentrati nel definire le caratteristiche peculiari degli agenti ed i loro comportamenti, piuttosto che nel modellare dettagli tecnici od implementativi.

Nella sezione II verranno presentati e discussi brevemente gli approcci attualmente utilizzati per lo sviluppo di applicazioni ad agenti. Nella sezione III verrà invece presentato un breve panorama dei linguaggi di programmazione orientati agli agenti descrivendone le caratteristiche principali. Nella sezione IV verranno illustrati gli aspetti caratteristici del linguaggio Agent# fornendo anche una breve descrizione della piattaforma AgentService e della Common Language Infrastructure (CLI) [5]. Un esempio applicativo dell'utilizzo di Agent# verrà presentato nella sezione V e quindi un confronto con i precedenti linguaggi di programmazione ad agenti nella

Articolo ricevuto il 30 Giugno 2003.

C. Vecchiola collabora con il Laboratorio di Informatica DIST Siemens-Orsi, Università di Genova, Genova, GE 16145 Italia, (e-mail: capsules@email.it).

M. Coccoli è affiliato al Dipartimento di Informatica, Sistemistica, Telematica, Università di Genova, Genova, GE 16145 Italia, (phone: +39-10-3532284; fax: +39-10-3532154; e-mail: coccoli@dist.unige.it).

A. Boccalatte è affiliato al Dipartimento di Informatica, Sistemistica, Telematica, Università di Genova, Genova, GE 16145 Italia (e-mail: nino@dist.unige.it).

sezione VI. Infine alcune riflessioni sui futuri sviluppi del lavoro qui presentato saranno discussi nella sezione VII.

## II. APPROCCI PER LA PROGRAMMAZIONE AD AGENTI

Lo sviluppo di sistemi ad agenti è stato affrontato nel tempo seguendo diversi approcci. Si è cercato infatti di costruire modelli che permettessero di definire delle specifiche formali del comportamento di sistemi ad agenti, ed allo stesso tempo sono stati forniti strumenti che permettessero la programmazione dei sistemi multi-agente. La realizzazione di specifiche formali di sistemi multi-agente è stata affrontata utilizzando modelli di programmazione logica di diversa natura [6]. Gli strumenti per la programmazione dei sistemi multi-agente sono stati realizzati come pacchetti software sviluppati utilizzando la programmazione orientata agli oggetti [7]. Questi pacchetti software sono normalmente conosciuti come framework per la programmazione ad agenti.

L'approccio basato sulla programmazione logica è maggiormente orientato alla definizione di un comportamento per l'agente e delle modalità di interazione che i diversi agenti instaurano; sono state sviluppate architetture che modellano la conoscenza che un agente ha del mondo, le sue intenzioni e gli obiettivi che esso intende perseguire [8] che utilizzano come strumenti la logica dei predicati del primo ordine, la logica multimodale o altre soluzioni che costituiscono delle varianti a questi modelli. Soluzioni di questo tipo sono molto efficaci a livello espressivo ma spesso non permettono un'integrazione del tutto naturale con altri modelli di sviluppo software.

L'approccio proposto con i framework per la programmazione di agenti, risulta in tal senso sicuramente più versatile ed è quello maggiormente utilizzato per lo sviluppo di sistemi multi-agente reali. Normalmente un framework per lo sviluppo di sistemi multi-agente propone un proprio modello di agente e di sistema multi-agente, offrendo una serie di strumenti e di interfacce semplificate che implementano processi automatizzati che guidano lo sviluppatore nella corretta definizione del sistema e degli agenti. Queste sono soluzioni che aumentano la produttività, ma molto spesso le possibilità di intervento dello sviluppatore sono limitate sacrificando a volte l'efficacia di questo approccio. Spesso i sistemi multi-agente sviluppati attraverso i framework si appoggiano a una gerarchia di classi di un qualche linguaggio orientato agli oggetti (normalmente Java o C++) accessibile allo sviluppatore software. L'utilizzo diretto della gerarchia di classi offre sicuramente maggiori possibilità, ma si corre il rischio di perdere la prospettiva orientata agli agenti nello sviluppo del codice, soffermandosi su aspetti più tecnici della progettazione software, come ad esempio la gestione del threading. Inoltre seguendo questa strada spesso occorre ripensare in termini di oggetti l'applicazione originariamente progettata in termini di agenti.

L'approccio basato sulla programmazione logica risulta adatto a definire l'intelligenza di un agente ma sicuramente non un agente software nel suo complesso, mentre l'approccio proposto con i framework offre un modello che permette di modellare efficacemente un sistema multi-agente ma può portare fuoristrada lo sviluppatore. La soluzione che prevede

l'impiego di framework è sicuramente la più vantaggiosa in quanto è possibile sia trovare delle direttive di implementazione [9], sia integrare nell'architettura del sistema anche la soluzione proposta dalla programmazione logica.

## III. LINGUAGGI DI PROGRAMMAZIONE ORIENTATI AGLI AGENTI

Il primo approccio allo sviluppo di applicazioni orientate agli agenti utilizzando un linguaggio di programmazione risale al 1991 con il linguaggio di programmazione Agent-0 [10]. I linguaggi di programmazione ad agenti permettono agli sviluppatori di definire e programmare agenti in accordo con un particolare modello di agente (quello supportato dal linguaggio). L'architettura BDI [11] è il modello maggiormente implementato da questi linguaggi di programmazione. Tali linguaggi normalmente modellano gli agenti come componenti di un MAS, o come un insieme di asserzioni e clausole che debbono essere processate da un opportuno interprete. Sono stati perciò presi come modello per sviluppare i linguaggi di programmazione ad agenti due paradigmi di programmazione: il modello orientato agli oggetti e la programmazione logica. In particolare, la tecnologia Java [12] è stata adottata come base per tutti quei linguaggi che hanno scelto il modello orientato agli oggetti come riferimento. Verrà ora presentata una selezione dei differenti linguaggi di programmazione orientati agli agenti, tali linguaggi verranno successivamente presi come riferimento per discutere le caratteristiche di Agent#.

### A. Agent-0, Placa Extensions, Agent-K

I primi passi nel campo dei linguaggi di programmazione orientati agli agenti sono stati caratterizzati dall'impiego di linguaggi formali e dalla loro traduzione in linguaggi di programmazione. Un nutrito interesse si è sviluppato verso la definizione della semantica di questi linguaggi formali e tali linguaggi hanno portato allo sviluppo di Agent-0 e delle sue successive estensioni: PLACA [13] ed Agent-K [14].

Agent-0 è il primo linguaggio di programmazione orientato agli agenti. Agent-0 permette di modellare un agente definendone *beliefs*, *obligation*, e *capabilities*. Con il termine *belief* si identificano asserzioni che descrivono proprietà dell'ambiente in un determinato istante di tempo. Una *obligation* indica invece l'impegno di rendere vera un'asserzione in un determinato istante di tempo. Le *capabilities* sono asserzioni sulle capacità di un agente. In Agent-0 il ciclo di vita di un agente è caratterizzato da un loop di controllo comandato dall'interprete del linguaggio: ciclicamente lo stato di un agente viene aggiornato con i messaggi ricevuti, e si eseguono le *obligations* utilizzando le *capabilities*.

Una delle limitazioni di Agent-0 è costituita dal fatto di non avere meccanismi per introdurre nuovi goal: le estensioni apportate con PLACA tentano di sopperire a questa mancanza introducendo nuovi elementi per manipolare le intenzioni di un agente.

Agent-K è un tentativo di standardizzare le funzionalità di comunicazione in Agent-0. Agent-K integra nella sintassi di Agent-0, KQML (Knowledge Query Manipulation Language) [26] permettendo così la gestione di messaggi realizzati in in altri linguaggi.

#### B. APRIL

APRIL (Agent Process Interaction Language) [16] è un linguaggio di programmazione orientato allo sviluppo di sistemi multi-agente, realizzato utilizzando come punto di partenza il C. APRIL è un linguaggio di programmazione orientato agli oggetti dove gli oggetti modellano dei processi: permette perciò la definizione di processi e dispone di strumenti che permettono la comunicazione tra differenti processi in un ambiente distribuito.

#### C. AgentSpeak(L)

AgentSpeak(L) [17] è un linguaggio di programmazione orientato agli agenti modellato per interagire con i sistemi PRS [18] e dMARS [19]. In AgentSpeak(L) la specifica di un agente avviene descrivendo una sequenza di clausole simili alle clausole di Horn [20]. AgentSpeak(L) è modellato come un linguaggio dei predicati del primo ordine con azioni ed eventi, e fornisce un'astrazione, implementando l'architettura BDI per gli agenti, dei sistemi su cui vengono implementati gli agenti.

#### D. JACK

JACK [21] è un framework multi-agente che estende il linguaggio Java. Gli elementi costitutivi di JACK sono: un insieme di librerie di classi che costituiscono il nucleo del framework multi-agente; un insieme di estensioni al linguaggio Java ed un compilatore opportunamente modificato. JACK è stato progettato per supportare diversi modelli di agente come quello proposto dall'architettura BDI od altri. JACK è interamente integrato con la tecnologia Java e permette di utilizzare qualsiasi componente Java per definire la logica di programmazione di un agente.

#### E. APL

APL (Agent Programming Language) [22] è un linguaggio di programmazione orientato agli agenti per i quali implementa l'architettura BDI. APL fornisce dei template per ogni componente del modello BDI ed utilizza una sintassi molto simile a quella del linguaggio Java. APL è stato modellato avendo come base la tecnologia Java ed ogni template definito viene tradotto in una classe Java dal compilatore APL. Il linguaggio offre inoltre una persistenza automatica per le beliefs di un agente e la possibilità per gli agenti di cambiare i propri obiettivi a run-time.

### IV. AGENT#

#### A. Caratteristiche di progetto

Agent# è stato pensato per programmare agenti software che operano all'interno della piattaforma AgentService, che definisce così il contesto delle applicazioni realizzate con il linguaggio. Sia la piattaforma AgentService che il linguaggio

Agent# sono stati realizzati sfruttando le potenzialità rese disponibili dalla tecnologia .NET, in quanto AgentService è una piattaforma ad agenti interamente realizzata in C# [23], mentre il compilatore per il linguaggio Agent# ed il linguaggio stesso sono stati progettati per realizzare un codice oggetto eseguibile nel run-time di .NET e per sfruttare le funzionalità che questo mette a disposizione.

Agent# espone il modello di agente proposto dalla piattaforma AgentService attraverso dei template programmabili che costituiscono gli unici elementi definibili all'interno del linguaggio. In questo modo anche la produzione del codice mantiene una prospettiva orientata agli agenti. Il linguaggio presenta inoltre dei costrutti che permettono un accesso naturale alle risorse condivise garantendo allo stesso tempo la correttezza e la consistenza dei dati contenuti nella base di conoscenza dell'agente. Queste caratteristiche sono state elegantemente realizzate sfruttando alcuni elementi innovativi della tecnologia .NET ed in particolare della Common Language Infrastructure.

La programmazione orientata agli agenti ha naturalmente a che fare con la rappresentazione e l'interazione con sistemi complessi; fino ad oggi lo strumento più potente per modellare e rappresentare sistemi complessi è stata la programmazione orientata agli oggetti [24], per tal motivo la sintassi e la semantica dei linguaggi orientati agli oggetti è stata presa come elemento base per lo sviluppo di Agent#. Più precisamente è stato preso a modello, per la sintassi e la semantica, il linguaggio C#. Il linguaggio Agent# ha infatti mantenuto praticamente inalterate sia la sintassi che la semantica dei costrutti C# per quanto riguarda la grammatica delle espressioni e degli statements che costituiscono il nucleo per la codifica degli algoritmi, mentre ha significativamente modificato i costrutti di alto livello del C#, eliminando il costrutto class ed interface, ed introducendo i costrutti di alto livello propri della programmazione orientata agli agenti

#### B. AgentService e la Common Language Infrastructure (CLI)

Il linguaggio di programmazione Agent# si appoggia alla piattaforma AgentService per realizzare applicazioni orientate agli agenti. Il modello di sistema multi-agente implementato nel pacchetto software AgentService è infatti il target degli agenti realizzati con Agent#.

AgentService offre un framework di programmazione ad agenti basato sulla Common Language Infrastructure e modellato e modellato sulle specifiche architetturali promosse da FIPA. L'utilizzo della Common Language Infrastructure come infrastruttura per lo sviluppo dell'intero pacchetto software AgentService, ha permesso al framework di sfruttare le caratteristiche innovative di questa tecnologia: tra queste ricordiamo l'interoperabilità tra i diversi linguaggi basati sulla CLI e gli Application Domain. Sebbene AgentService sia stata interamente realizzata in C#, gli agenti o gli altri componenti che la piattaforma gestisce possono essere implementati in altri linguaggi di programmazione come ad esempio Visual Basic .NET, Visual C++.NET, Oberon e quindi anche Agent#. Gli Application Domain sono stati invece un elemento fondamentale per garantire il corretto isolamento di un agente software, senza limitarne le funzionalità. La CLI offre altre

funzionalità oltre a quelle presentate come ad esempio il fatto di offrire un ambiente di run-time portabile tra le diverse piattaforme hardware e software.

La libreria AgentService fornisce allo sviluppatore software le seguenti funzionalità:

- definizione di agenti autonomi e persistenti;
- esecuzione concorrente degli agenti dei loro diversi comportamenti;
- strutture dati persistenti e condivise all'interno del singolo agente;
- modello di comunicazione transazionale basato sullo scambio di messaggi;
- accesso ai componenti di servizio FIPA.

Il modello strutturale per un agente proposto da AgentService, considera un agente come un'entità software il cui stato è definito da un insieme di dati chiamati Knowledge e le cui attività sono modellate con oggetti chiamati Behavior. L'esecuzione concorrente dei Behavior che compongono l'agente e l'accesso controllato allo stato dell'agente tramite le Knowledge, permette di avere un modello di esecuzione realmente concorrente ed allo stesso tempo sicuro.

AgentService fornisce inoltre servizi di comunicazione, di pagine gialle e bianche, nonché un insieme di tool amministrativi per il controllo delle istanze della piattaforma.

### C. Iter di sviluppo di un'applicazione con Agent#

Volendo sviluppare un'applicazione realizzata con il linguaggio Agent#, potremmo individuare il seguente iter:

- diversi componenti vengono progettati e realizzati nei linguaggi supportati dalla CLI;
- gli agenti vengono realizzati in Agent# e vengono integrati componenti precedentemente progettati;
- il codice prodotto viene compilato in librerie di codice gestito (managed code);
- le librerie così realizzate possono essere caricate come moduli all'interno della piattaforma AgentService;
- gli agenti realizzati possono essere utilizzati dinamicamente qualora se ne faccia richiesta alla piattaforma.

L'interoperabilità tra i diversi linguaggi di programmazione supportati dalla Common Language Infrastructure gioca un ruolo fondamentale nello sviluppo di applicazioni con Agent#. Come mostra la figura IV.1 è infatti possibile integrare all'interno dei template offerti da Agent#, e quindi utilizzare, componenti sviluppati in altri linguaggi di programmazione.

Sebbene Agent# disponga di una grammatica per le espressioni e gli statement molto ricca, è stato principalmente pensato per sviluppare e modellare gli aspetti orientati agli agenti di un progetto software e per delegare ad altri linguaggi di programmazione la progettazione e la realizzazione di componenti comunque utili ma non strettamente legati alla teoria degli agenti. La possibilità di poter istanziare ed utilizzare qualsiasi componente progettato per la CLI all'interno di Agent# conferisce a tale linguaggio di programmazione la possibilità di avere a disposizione una vasta gamma di risorse direttamente utilizzabili nelle proprie

applicazioni. Questa particolare caratteristica di progetto permette inoltre di dividere più agevolmente il lavoro per competenze.

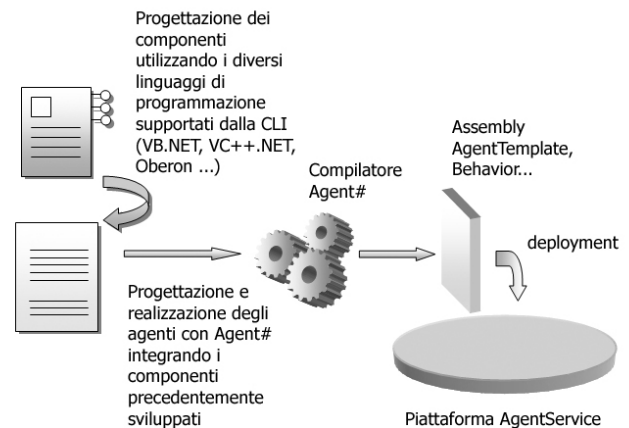


Figura IV.1 Modello di sviluppo di un'applicazione con Agent#

Per quanto riguarda lo sviluppo strettamente correlato all'utilizzo di Agent# il linguaggio offre tre template che modellano i concetti chiave dell'architettura ad agenti adottata. Tali template modellano il concetto di: agente, knowledge e behavior e verranno discussi in maggior dettaglio nelle sezioni che seguono.

### D. Il Template agent

Il template per la definizione di differenti tipi di agente permette di definire un agente specificandone i componenti che lo caratterizzano: tali componenti possono essere dei comportamenti o delle istanze di Knowledge. E' inoltre possibile definire del codice di inizializzazione e di clean-up dell'agente per permettere così che le istanze dei singoli agenti vengano correttamente costruite dalla piattaforma e terminate in modo da rilasciare tutte le risorse.

In figura IV.2 è riportata una possibile definizione di un agente: il template è strutturato in quattro sezioni in cui vengono dichiarate le knowledge, i comportamenti, opportunamente configurati con le knowledge che devono utilizzare, il codice di inizializzazione e quello di clean-up.

```
agent TemplateName {
  shared { // dichiarazione knowledge
    KnowledgeTemplate1 _k1;
    KnowledgeTemplate2 _k2;
  }
  behaviors { // dichiarazione behavior
    BehaviorTemplate1 _b1[_k1, _k2];
    BehaviorTemplate2 _b2[_k2];
    BehaviorTemplate3 _b3[_k1];
  }
  init { // codice di inizializzazione
  }
  done { // codice di clean-up
  }
}
```

Figura IV.2. Definizione di un agente

Come possiamo notare in figura IV.2, il codice che esprime le attività dell'agente non è localizzato all'interno della definizione dell'agente: questo è infatti un contenitore opportunamente configurato con particolari componenti. Il codice che esprime il comportamento aggregato dell'agente è localizzato nei differenti behavior di cui è composto e questo permette un elevato grado di riusabilità delle funzionalità già progettate dando la possibilità di definire altri tipi di agenti componendo comportamenti già progettati.

#### E. Il Template knowledge e lo Statement freeze

Il template knowledge modella un insieme di dati correlati che devono essere soggetti a controllo di accesso in quanto condivisi tra le diverse attività che compongono un agente.

```
knowledge KnowledgeTemplate1 {
    System.DateTime arrivalTime;
    int orderNumber;
    // ogni oggetto supportato dalla
    // CLI eccetto delegate ed eventi
}
```

Figura IV.3 Definizione di una knowledge

La figura IV.3 mostra una possibile definizione di knowledge: come possiamo vedere questa è molto simile ad un record del linguaggio Pascal [25]. Una knowledge è identificata da un nome ed i tipi dei campi che contiene sono tutti quelli supportati della Common Language Infrastructure. L'accesso ai campi di un'istanza di un qualunque knowledge template avviene come nel caso dei membri di un oggetto. Poiché queste strutture dati sono sottoposte al controllo d'accesso, l'accesso a tali campi può avvenire solo all'interno di un contesto sicuro: tale contesto è individuato dallo scope del blocco introdotto dalla parola riservata `freeze`. Questo è uno statement molto simile allo statement `lock` del linguaggio C# od allo statement `synchronized` presente nel linguaggio Java, ma opportunamente realizzato per la gestione delle knowledge. Per poter utilizzare gli oggetti knowledge lo sviluppatore deve aprire un blocco `freeze` dichiarando la lista delle istanze knowledge a cui intendere accedere. All'interno del blocco `freeze` le istanze dichiarate sono acquisite in modo esclusivo e si ha un accesso sicuro; al di fuori di questo blocco le istanze non hanno visibilità. In questo modo la consistenza dei dati condivisi è garantita a livello di design e non deve essere programmata dallo sviluppatore.

#### F. Il Template behavior

Un behavior modella un comportamento od un'attività di un agente: questa può essere una funzionalità od un'attività orientata a perseguire un obiettivo. I behavior vengono utilizzati per implementare algoritmi, servizi, o particolari protocolli: per tale motivo costituiscono gli elementi in `Agent#` in cui viene sviluppato il codice delle applicazioni orientate agli agenti. Il template behavior è stato progettato per essere un componente riutilizzabile all'interno di un `agent template` con il minimo impatto. Tali elementi infatti, sono accoppiati alle

knowledge che usano e che costituiscono le interfacce per scambiare i dati con un behavior. Per utilizzare un behavior all'interno di un `agent template` è perciò sufficiente dichiarare un'istanza del behavior e delle knowledge che richiede associandole a questo.

```
behavior BehaviorTemplate1 {
    shared { // dichiarazione knowledge
        KnowledgeTemplate1 _k1;
        KnowledgeTemplate2 _k2;
    }
    fields {
        System.DateTime arrivalTime;
        int order;
    }
    methods {
        void compute(System.DateTime dt) {...}
    }
    body { // entry-point
    }
}
```

Figura IV.4 Definizione di un behavior

La figura IV.4 mostra la struttura del template per definire un behavior. La sezione `shared` contiene tutte le istanze delle knowledge che il behavior utilizza. Le sezioni `fields` e `methods` contengono rispettivamente gli attributi ed i metodi che definiscono il behavior, mentre il blocco `body` costituisce l'entry point per l'esecuzione del behavior in quanto a run time la computazione associata al behavior avrà inizio mandando in esecuzione il codice contenuto in questo blocco.

Il template Behavior è molto simile ad una classe anche se in esso non è presente la nozione di ereditarietà: il corpo dei metodi infatti presenta sostanzialmente le stesse possibilità espressive di un generico metodo definibile all'interno di una classe in C#. Il programmatore può perciò decidere se esprimere algoritmi anche complessi all'interno del behavior oppure se delegare tale compito a componenti sviluppati in altri linguaggi ed usarne le funzionalità.

## V. UN CASO DI STUDIO

Per illustrare la metodologia di sviluppo di un'applicazione con `Agent#` ed `AgentService`, verrà modellato il problema della pianificazione delle traiettorie di più robot che si trovano in uno stesso ambiente. Occorre in tal caso fare in modo che i differenti robot evitino gli ostacoli fissi e mobili svolgendo il compito che gli è stato assegnato.

Il problema può essere simulato e studiato creando un sistema multi-agente in cui ogni robot è rappresentato da un agente. I diversi robot possono essere modellati in `Agent#` con lo stesso template in quanto devono tutti effettuare le stesse operazioni, sebbene differiscano nelle condizioni iniziali e finali. Per definire il comportamento aggregato del template che modella il robot occorre individuare le funzionalità base che questo deve offrire e quindi progettare i differenti behavior che svolgono tali compiti associati alle opportune knowledge.

Lo stato del robot può essere definito considerando le informazioni che questo necessita per svolgere i propri compiti.

Una volta individuate tali informazioni queste verranno opportunamente partizionate definendo i template per le knowledge da associare ai behavior. Sarà perciò necessario disporre di una knowledge che contenga la traiettoria pianificata, la posizione e la velocità correnti del robot. Saranno inoltre necessarie una knowledge che contenga lo stato finale desiderato per il robot, ed una knowledge in cui viene mantenuta la mappa dell'ambiente con i diversi ostacoli aggiornata ad ogni iterazione.

Possiamo pensare di scomporre le funzionalità del robot in tre behavior che chiameremo operator, planner, e communicator. Il behavior operator gestisce la sensoristica e gli attuatori del robot simulato, ed interagisce con le knowledge che contengono lo stato interno del robot e la mappa dell'ambiente aggiornandole. Il planner pianifica la traiettoria on-line e necessita di interagire con tutte le knowledge definite, mentre il communicator comunica agli altri robot la propria posizione e aggiorna la mappa dell'ambiente con le informazioni che riceve dagli altri robot.

Il problema descritto è stato facilmente scomposto utilizzando i template offerti Agent#, inoltre la scomposizione operata favorisce il riuso: è possibile cambiando i behavior definire agenti-robot con diversi algoritmi di pianificazione o diverso hardware simulato. Alcuni dei behavior definiti possono inoltre essere impiegati per realizzare agenti che non simulino robot ma che ad esempio richiedano funzionalità di pianificazione.

## VI. CONFRONTO

I linguaggi che presentano maggiori somiglianze con Agent# sono APL e JACK: APL è fortemente basato sull'architettura BDI per realizzare agenti mentre JACK offre agli sviluppatori diversi modelli, tra cui anche l'architettura BDI. Agent# non offer un supporto nativo all'architettura BDI sebbene questa possa essere facilmente modellata utilizzando knowledge e behavior.

In APL i programmatori sono legati al modello BDI e gli elementi stessi del linguaggio come variabili e metodi sono direttamente mappati su beliefs ed intentions. In Agent# gli sviluppatori sono liberi di organizzare behavior e knowledge come preferiscono eventualmente implementando anche un'architettura BDI. APL offre persistenza trasparente a tutte le beliefs, in Agent# la persistenza delle knowledge non è gestita in quanto è la piattaforma AgentService stessa ad effettuare la persistenza automatica delle knowledge degli agenti presenti nel sistema. APL è un linguaggio di programmazione con tipi di prima classe: tutti gli elementi che modellano le componenti dell'architettura BDI possono essere usati come gli altri tipi e quindi essere soggetti, ad esempio, ad assegnamento. Agent# non presenta questa caratteristica in quanto offre tipi definibili ma non manipolabili: questo non è di fatto limitativo in quanto i tipi definiti sono automaticamente gestiti dalla piattaforma a run-time.

Le funzionalità offerte da JACK sono molto simili a quelle

offerte da Agent# ed AgentService: come nel caso di JACK infatti Agent# offre un linguaggio costruito come estensione di un altro per introdurre una prospettiva maggiormente orientata agli agenti. Sia JACK che Agent# sono completamente integrati con una tecnologia che offre una vasta gamma di risorse per la programmazione delle applicazioni. Attualmente Agent# offre un modello meno ricco di caratteristiche rispetto a quello offerto da JACK, ma allo stesso tempo molto flessibile.

APRIL risulta sostanzialmente diverso da Agent# in quanto non è di fatto un linguaggio specificamente progettato per la programmazione ad agenti sebbene disponga di funzionalità che facilitano la realizzazione di sistemi multi-agente.

Per quanto riguarda invece il rapporto con Agent-0 e le sue evoluzioni PLACA ed Agent-K, possiamo osservare che l'approccio adottato con Agent# è significativamente diverso. Agent# è stato sviluppato a partire da un linguaggio di programmazione general purpose, non come traduzione di una specifica formale.

## VII. CONCLUSIONI

In questo articolo è stato presentato Agent# un linguaggio di programmazione orientato agli agenti basato sulla tecnologia .NET. Agent# permette di realizzare agenti per il framework di programmazione ad agenti AgentService, offrendo un ambiente di programmazione per essa maggiormente orientato agli agenti. La scelta di offrire un linguaggio di programmazione specifico come strumento per la programmazione degli agenti merita qualche ulteriore approfondimento, alla luce dell'ormai frequente utilizzo strumenti per la generazione automatica del codice sorgente.

Il maggior contributo dovuto ad un particolare linguaggio di programmazione è quello di conferire una ben chiara prospettiva da cui affrontare una determinata classe di problemi e facilitarne la risoluzione offrendo degli strumenti opportuni. Il particolare linguaggio supporta un determinato paradigma di programmazione e lo rende facilmente applicabile attraverso i costrutti nativi che esso mette a disposizione. In tal senso l'adozione del paradigma è cablata nel linguaggio. Una seconda caratteristica che occorre valutare è l'espressività; un linguaggio di programmazione ha sicuramente una capacità espressiva maggiore rispetto alle possibilità offerte da un framework.

La maggior espressività di un linguaggio di programmazione deriva sostanzialmente dal fatto che esso stesso, rispetto ad un framework, è uno strumento di più basso livello. Occorre infine considerare il fatto che la realizzazione di un linguaggio è comunque un compito impegnativo se si deve realizzare uno strumento professionale o comunque utilizzabile in modo intensivo; nel caso delle applicazioni orientate agli agenti tale sforzo risulta sicuramente giustificato in quanto la tecnologia orientata agli agenti sta avendo una sempre più larga diffusione.

Agent# è un progetto ancora in corso e nuove funzionalità verranno aggiunte, per ottenere una più completa integrazione con i servizi offerti dalla piattaforma AgentService.

## REFERENCES

- [1] G. Weiss, *Multi-agent Systems – A Modern Approach to Distributed Artificial Intelligence*, G. Weiss Ed., Cambridge, MA, 1999.
- [2] E. Oliveira, K. Fischer, and O. Stepankova, "Multi-agent systems: which research for which applications" *Robotics and Autonomous Systems*, vol. 27, Elsevier, North-Holland, pp. 91-106, 1999.
- [3] A. Grosso, M. Coccoli, A. Gozzi, A. Boccalatte "An Agent Programming Framework based on C# and the CLI", in *Proc. 1st International Workshop on C# and .NET Technologies on Algorithms, Computer Graphics, Visualization, Distributed and WEB Computing*, Plzen, Czech Republic, February 2003, pp. 13-20.
- [4] C. Mingins, N. Nicoloudis, ".NET: a new Component-oriented Programming Platform", *Journal of Object-oriented Programming*, October 2001
- [5] *Common Language Infrastructure (CLI) partition I-IV*, ECMA standard 335. Disponibile: <http://www.ecma-international.org/publications/standards/ECMA-335.HTML>
- [6] N. R. Jennings, M. Wooldridge, "Agent Theories, Architectures, and Languages: A Survey", in *Proc. ECAI94 Workshop on Agent Theories Architectures and Languages*, The Netherlands, 1994
- [7] P. M. Ricordel, Y. Demazeau, "From Analysis to Deployment : a Multi-Agent Platform Survey", in *Proc. 1st Int. Work. Engineering Societies in the Agents' World, ESAW'00*, Berlin, Germany, August 2000, pp. 93-105.
- [8] A.S. Rao and M.P. Georgeff, "Modeling rational agents within a BDI-architecture", in *Proc. 2<sup>nd</sup> International Conference on Principles of Knowledge Representation and Reasoning (KR'91)*, J. Allen, R. Fikes, and E. Sandewall, Eds., Morgan Kaufmann, pp. 473-484. 1991.
- [9] FIPA Abstract Architecture Specification, FIPA standard SC00001L, Disponibile: <http://www.fipa.org/specs/fipa00001/SC00001L.pdf>
- [10] Shoham, Y., "AGENT-0: A simple agent language and its interpreter", in *9<sup>th</sup> National Conference on Artificial Intelligence*, Vol II. Anaheim, CA: MIT Press. pp. 704 – 709, 1991
- [11] A. Rao, M. Georgeff, "BDI Agents from Theory to Practice", in *Proc. 1<sup>st</sup> Int. Conf. on Multi-Agent Systems (ICMAS-95)*, San Francisco, USA, April 1995.
- [12] K. Arnold, J. Gosling, *The Java Programming Language*, 2nd Ed., Addison-Wesley, January 1998.
- [13] S. R. Thomas, "The PLACA agent programming language", in *Intelligent Agents*, M. J. Wooldridge and N. R. Jennings, Eds, Springer Verlag, Berlin, 1995, pp. 355-370.
- [14] W. H. E. Davies, P. Edwards, Agent-K: An Integration of AOP & KQML, in *Proceedings of the CIKM '94 Workshop on Intelligent Information Agents*, Y. Labrou, T. Finin Eds., National Institute of Standards & Technology, Gaithersburg, Maryland, USA, 1994.
- [15] T. Finin, G. Wiederhold, "An Overview Of KQML: A Knowledge Query and Manipulation Language", Dept. Computer Science, Stanford University, Stanford, CA, 1991
- [16] F. McCabe, K. Clark, "April: Agent Process Interaction Language in Intelligent Agents", in *LNCS*, vol. 890, M. Wooldridge, N.R. Jennings, Eds, Springer-Verlag, 1995
- [17] A. S. Rao, "AgentSpeak(L): BDI agents speak out in a logical computable language", in *Agents Breaking Away: Proc. 7<sup>th</sup> European Workshop on Modelling Autonomous Agents in a Multi-Agent World*, LNAI vol 1038, Springer-Verlag, pp. 42-55, 1996.
- [18] M.P. Georgeff and F.F. Ingrand, "Decision – Making in an embedded reasoning system", in *Proc. 5<sup>th</sup> Int. Joint. Conf. Artificial Intelligence IJCAI*, Detroit, 1989
- [19] M. d'Inverno, D. Kinny, M. Luck, and Michael Wooldridge. "A formal specification of dMARS", in *Intelligent Agents IV: Proc. 4<sup>th</sup> International Workshop on Agent Theories, Architectures, and Languages*, LNAI 1365, M.P. Singh, A.S. Rao, and M. Wooldridge, Eds., Springer-Verlag pp. 155-176, 1998
- [20] A. Horn, "On Sentences Which Are True of Direct Unions of Algebras.", in *J. Symbolic Logic* no.16, pp. 14-21, 1951
- [21] P. Busetta, R. Ronquist, A. Hodgson, and A. Lucas, "JACK Intelligent Agents – Components for Intelligent Agents in Java", Agent Oriented Software Pty. Ltd, Melbourne, Australia, Tech. Rep.1,1998
- [22] Jo Chang-Hyun. and K. M. Geroge, Agent-based Programming Language: APL, In *Proc. 2002 ACM symposium on Applied computing*, Madrid, Spain, 2002, pp 27-31.
- [23] *C# Language Specifications*, ECMA standard 334. Disponibile: <http://www.ecma-international.org/publications/files/ecma-st/Ecma-334.pdf>
- [24] G. Booch, "Object-Oriented Development," *IEEE Trans. Software Eng.*, vol. SE-12, no. 2 pp. 211-221, Feb. 1986
- [25] N. Wirth, "The Programming Language Pascal", *Acta Informatica*, vol. 1, no. 1, pp. 35-63, 1971
- [26] T. Finin, G. Wiederhold, "An Overview Of KQML: A Knowledge Query and Manipulation Language", Dept. Computer Science, Stanford University, Stanford, CA, 1991