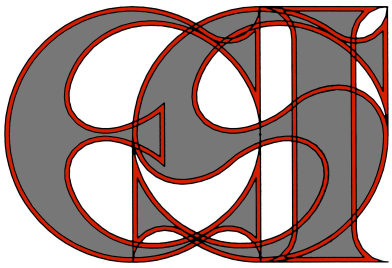

Descrizione del progetto

AUCTORES

26 Dicembre 2006

Autore: Sergio Iovacchini

Data: 13 marzo 2007

I N D I C E

Descrizione sommaria di AUCTORES 3
Motivazione del progetto 4
Caratteristiche di AUCTORES 6
Principi costitutivi 6
Operatori 6
Ambienti di conoscenza 7
Organizzazione e classificazione della conoscenza 7
I connettori 8
La funzionalità 9
Elaborazione parallela 10
Possibile definizione di un'algebra di composizione 10
Utilizzo di AUCTORES 12
CAD per la programmazione 12
Aspetto grafico 13
Altri metodi per la creazioni di operatori 13
Operatori Esterni: il Launcher 14
Gli obiettivi della ricerca 16
Nucleo di AUCTORES (OR1) 16
CAD per la programmazione (OR2) 17
Definizione di strutture algebriche (OR3) 17
Finalità del progetto 17
Tempi di realizzazione e risorse impiegate nel progetto 18
Conclusioni 19

Descrizione sommaria di AUCTORES.

Si tratta di un sistema di programmazione derivato dai principi della programmazione funzionale il cui scopo principale è quello di facilitare la programmazione, consentendo all'utente di realizzare l'applicazione desiderata componendo funzionalità già realizzate senza dover far ricorso a conoscenze proprie dell'informatica.

Il mezzo con cui ciò è reso possibile è l'utilizzo di meccanismi di automazione basati sulla classificazione, sull'utilizzo di strutture decisionali, e soprattutto sulla possibilità di riuso completo di quanto prodotto in precedenza.

Esso è basato sulla composizione di entità che elaborano le informazioni, dette ***operatori*** per analogia con quelli dell'algebra matriciale.

Particolare attenzione è posta all'efficienza di utilizzo in elaboratori con architettura vonNeumann, ed alla parallelizzazione automatica di processi in elaboratori multiprocessori quali per esempio i mainframes ed i processori CELL di IBM/Sony/Toshiba.

AUCTORES realizza un formalismo, interpretabile dall'hardware, per mezzo del quale si compie una *descrizione operativa* dei concetti.

Questa caratteristica potrebbe essere una base su cui fondare un sistema algebrico in modo che l'elaborazione consista essenzialmente nella valutazione di equivalenze formali e che richieda processi algoritmici soltanto nella fase finale.

Lo strumento principale di interazione con AUCTORES è un compositore grafico che si serve delle informazioni di classificazione degli *operatori* per guidare interattivamente l'utente.

Si può utilizzare AUCTORES anche per mezzo di linguaggi formali, per mezzo di strutture XML, per mezzo di interfacce ad hoc, per esempio per i disabili.

Un apposito ***launcher*** permette di eseguire, in modo autonomo, l'applicazione costruita con AUCTORES in tutti i sistemi operativi previsti.

Motivazione del progetto.

AUCTORES ha lo scopo di semplificare la programmazione per coloro che non intendono assoggettarsi a conoscenze specialistiche di informatica per poter produrre proprie applicazioni.

La conoscenza specialistica necessaria è contenuta nei componenti di AUCTORES e ad essa si può accedere per mezzo di interfacce specializzate, siano esse grafiche, orientate ad utenti menomati quali ipovedenti, paraplegici od altro; oppure anche per mezzo di interfacce basate su linguaggi formali o naturali etc...

Le motivazioni che stanno alla base delle scelte rivolte alla realizzazione del progetto AUCTORES derivano da considerazioni sugli strumenti che l'informatica oggi offre agli utenti di elaboratori.

Attualmente, nonostante la disponibilità di prodotti di supporto alla programmazione, il tempo necessario per produrre una propria applicazione è comunque notevole, ed inoltre richiede un livello di conoscenza legato ai principi ed allo strumento di programmazione che non è immediatamente raggiungibile da parte di utenti non specialisti di informatica.

L'introduzione della programmazione ad oggetti, oggi quasi universalmente alla base del software prodotto, esaspera ancora di più questo aspetto, poiché, mentre esso si dimostra funzionale per gli sviluppatori professionisti, richiede da parte di altri conoscenze ancora più specifiche ed anche un approccio logico specifico.

Altra caratteristica degli attuali sistemi e strumenti di programmazione è il fatto che un'applicazione sia strettamente collegata all'ambiente di fruizione grafica in cui essa è "annegata" e questo nonostante si siano fatti sforzi per offrire un "ambiente grafico di interazione" che si interponga tra il programma ed il sistema ospite. Ciò comporta sia il fatto di dover strutturare l'applicazione stessa in funzione dell'interazione con l'utente, sia, a causa di ciò, il dover spendere risorse nella programmazione dell'interfaccia.

Queste considerazioni valgono non soltanto per quanto riguarda gli strumenti di programmazione a "basso livello", ma purtroppo anche per i linguaggi di scripting.

Quindi, poiché non vengono attualmente offerti all'utente generico strumenti adeguati di programmazione di base, non avendo questi, nelle condizioni citate precedentemente, la possibilità di procedere autonomamente alla costruzione di propri strumenti di elaborazione, deve considerare l'utilizzo dei packages per poter compiere le elaborazioni desiderate.

L'utilizzo di packeges però implica che l'utente non può scegliere le funzionalità di cui ha bisogno, ma deve scegliere, se possibile, quale applicazione meglio si adatta alle sue esigenze, altrimenti deve utilizzare più packages (se non anche più elaboratori); inoltre il costo della gestione delle stesse informazioni, in modalità ed in ambiti diversi, è molto elevato in termini di tempo e denaro nonostante l'uso di database centralizzati e di formati di scambio informazioni sempre più universali.

Inoltre le diverse applicazioni disponibili non sono progettate per essere componibili tra loro; i loro progettisti, per considerare di quali funzionalità esse debbano venire dotate, ipotizzano degli "utenti tipo" e di fatto l'applicazione costruita segue così le modilità di svolgimento del lavoro di questo "utente" tipo"con le attività previste per lo specifico profilo.

Di fatto però spesso non esistono, soprattutto per l'utilizzo di elaborazioni non ripetitive, ruoli così ben definiti come i progettisti delle applicazioni software hanno ipotizzato.

Ciò comporta la "non componibilità" del software (sempre considerando, con le dovute eccezioni, diversi gradi di compobilità): l'utente non può, in genere, facilmente comporre le funzionalità in modo diverso da ciò che è stato previsto, ma può soltanto scegliere tra operazioni già previste.

Anche l'adattamento delle applicazioni per le quali si dispone del codice sorgente non è assolutamente agevole per chi non sia un professionista: l'uso di strutture di informazione globali e la stretta

dipendenza funzionale tra le varie parti dell'applicazione costituiscono un forte ostacolo per chi non sia esperto del software stesso.

Il progetto AUCTORES si propone di ricoprire il "buco" esistente tra i sistemi di elaborazione attuali; esso vuole essere una giusta via di mezzo tra il doversi realizzare una applicazione per mezzo di linguaggi di programmazione (inclusi i linguaggi di scripting) ed il doversi servire di packages. AUCTORES, per superare queste difficoltà, si rivolge all'esperienza maturata nel campo della programmazione funzionale e logica come base; considerando però anche quanto maturato nei sistemi software convenzionali, cioè gestione relazionale e gerarchica delle basi di dati, esperienze ergonomiche nell'interazione programma-utente, efficienza del codice oggetto. Esso quindi vuole essere uno strumento in grado di consentire di costruire propri strumenti di elaborazione secondo le seguenti modalità:

- raggiungimento della funzionalità desiderata, senza doversi preoccupare di sviluppare funzioni accessorie quali ad es. quelle di interazione;
- uso della *composizione* di software già realizzato per ottenere lo strumento per realizzare le **funzionalità desiderate in modo svincolato da particolari applicazioni**;
- utilizzo di **prototipi *by example*** per la costruzione dello strumento di elaborazione;
- possibilità di interazione per mezzo di **interfacce** non predeterminate, (colloquio basato su suoni, controllo di dispositivi tattili, etc...) che possano, per esempio, permettere anche a disabili di utilizzare sia lo strumento finale che la costruzione di strumenti;
- possibilità di fornire al costruttore di strumenti informazioni accessorie che permettano la costruzione di nuovi strumenti "senza dover ricominciare tutto da capo", cioè fornire un sistema in grado di **apprendere** ed **estendere** le proprie funzionalità.

Caratteristiche di AUCTORES.

Per ottenere lo scopo prefissato è necessario costruire un sistema autoconsistente, indipendente da qualsiasi tipo di applicazione.

Per garantire la flessibilità desiderata AUCTORES è un sistema di composizione di funzionalità in cui:

- ciascuna funzione è manovrabile da altre funzioni,
- tutte le funzioni sono tra loro componibili,
- ogni funzione realizza uno ed un solo scopo ben determinato,
- le operazioni, oltre che potersi riferire a diversi tipi di operandi, possono essere effettuate su insiemi (matrici, liste, alberi), dati con una propria struttura come frames, dati elementari, etc...,
- il polimorfismo per le funzioni si spinge solo sino a che valgano le stesse proprietà per le varie operazioni.

Principi costitutivi.

Per consuetudine, nel campo dell'informatica, l'elaborazione è considerata costituita da ***operazioni*** e ***dati***.

Questa divisione, pur essendo stata utile, soprattutto per il fatto di tenere il programma residente in memoria al riparo da modifiche, è fuorviante perchè anche i *dati* più banali presuppongono delle conoscenze implicite: occorre sapere in modo autonomo come elaborarli, per esempio i numeri sono tali solo se oggetto di operazioni matematiche.

È stato introdotto il concetto di **tipo** proprio per ovviare parzialmente a questa carenza di informazioni.

Si considerano quindi non *elaborazione* e *dati*, ma operazioni che sono compiute in determinati ambiti; in altri termini si esplicita il concetto di *tipo* specificando quali debbano essere le operazioni attese.

Secondo tale considerazione la rappresentazione di ogni entità è effettuata da una ***descrizione operativa***, che contiene al suo interno le operazioni che un ***fruitore*** di tale descrizione è in grado di interpretare: l'esistenza di una qualsiasi descrizione acquista significato per il fatto di essere utilizzata da un'altra entità.

Operatori.

La *descrizione operativa* è la descrizione che avviene a partire dalla composizione di concetti più elementari, ossia ***primitivi*** oppure già definiti, e corrispondenti ciascuno ad un'operazione. La rappresentazione di una *operazione* è detta ***operatore*** e, poiché una entità che *fruisce* di informazioni *fornite* da un'altra è a sua volta *fornitrice* di informazioni ad un'entità successiva nella catena del processo, ognuna di queste entità è un *operatore*.

Nella trattazione in corso, ciascun *operatore* è rappresentato da un ***simbolo***: un *simbolo* è un insieme (una configurazione) di rappresentazioni di **variazioni di stato (*bit*) su un supporto** quale esso sia, (per es. memoria); si intende per ***programma*** una ***sequenza di simboli***, mentre si intende per ***processo*** l'elaborazione determinata dal *programma* che lo descrive.

L'*operatore* così concepito assume significato soltanto in ***composizione*** con un altro *operatore* e l'insieme dei due costituisce il "processo minimo" che si può definire ed i due simboli corrispondenti sono il "programma minimo".

La *composizione* tra *operatori* consiste nel ***fornire*** da parte di un *operatore* (perciò detto ***operatore fornitore***) un ***simbolo*** all'*operatore* a cui è rivolta la *descrizione operativa*, nella forma e nelle modalità che hanno significato per esso (tale operatore è perciò detto ***operatore fruitore***).

Un *processo* è costituito da ***funzioni***, ciascuna di esse è rappresentata da un *operatore,* ed implica la composizione delle operazioni di cui è formato: l'elaborazione avviene in modo che il risultato di una prima *composizione* sia a sua volta oggetto di *composizione* con l'*operatore* logicamente successivo fino all'esaurimento del *processo* stesso: questo si riflette nel programma che lo rappresenta dove la composizione tra gli *operatori* (*simboli*) determina la produzione di un nuovo *simbolo* (*operatore*) che rappresenta a sua volta la composizione delle operazioni.

In un elaboratore del tipo di von Neumann, l'ultimo anello della catena di composizioni è la rappresentazione dei *simboli* in memoria; questi *simboli* sono elaborati da *operatori* che hanno il compito di trasformare la rappresentazione in memoria di tali *simboli* in *operatori* più complessi che saranno a loro volta composti con *operatori* di livello logico superiore.

Ambienti di conoscenza.

Poiché i sistemi di una certa complessità hanno bisogno di molte informazioni per poter funzionare, AUCTORES introduce il concetto di ***ambiente di conoscenza***, che è sia la collezione di informazioni necessarie alla elaborazione di gruppi omogenei di operazioni, sia il gestore della *fruizione* di queste informazioni da parte degli operatori con cui interagisce.

L'*ambiente di conoscenza* è ovviamente un operatore e, tranne situazioni eccezionali, a sua volta composto di *operatori*; un *ambiente di conoscenza*, in genere, è parte di un *ambiente di conoscenza* di livello superiore e composto da *ambienti di conoscenza*.

Le informazioni che sono contenute in essi sono ***descrizioni operative***, cioè composizioni di *operatori* che forniscono le informazioni all'operatore con cui l'*ambiente di conoscenza* si combina.

Esiste un ambiente di conoscenza universale che "racchiude" e gestisce tutto il sistema AUCTORES.

Organizzazione e classificazione della conoscenza.

Per poter favorire l'accesso alla programmazione a chi non utilizzi competenze specifiche in informatica, la descrizione della conoscenza, per AUCTORES, è organizzata in livelli per mezzo degli *ambienti di conoscenza*: al livello più elevato la descrizione delle funzionalità si compie usando i concetti e le strutture logiche proprie del compilatore umano; tali concetti e strutture sono a loro volta definiti in termini di concetti e strutture più elementari fino alle istruzioni interpretate dall' hardware.

I livelli più in basso sono quelli legati alle conoscenze relative all'hardware ed i cui *operatori* sono quelli che usano le funzioni di gestione hardware come primitive; le informazioni necessarie alla loro funzionalità sono organizzate in modo da contenere entro questi livelli le competenze informatiche.

Tra le informazioni che fanno parte dell'*ambiente di conoscenza* sono contenute informazioni di classificazione degli *operatori* e queste ultime sono utilizzate per fornire al *fruitore* le informazioni necessarie.

La classificazione consiste ovviamente nel prevedere l'uso delle informazioni, quindi ogni *operatore* è classificato in base agli *operatori* con cui esso potrebbe interagire.

Poiché però potrebbero (e questa è la normale situazione) costituirsi sempre nuovi *operatori*, non è possibile effettuare direttamente quanto affermato, ossia non è possibile prevedere tutte le possibili combinazioni tra *operatori*; per ovviare a ciò si considera una operazione come composta da un'**operazione di trasformazione** ed una di **collegamento** all'*ambiente di conoscenza*: gli *operatori* di

collegamento, cioè quelli che interagiscono direttamente con l'*ambiente di conoscenza*, sono detti ***connettori***.

I connettori.

I *connettori* sono collegati al concetto di ***tipo***, perciò il risultato della composizione di un *ambiente di conoscenza* con essi può essere precostituito: alla coppia *connettore-ambiente di conoscenza* è sostituita la risposta specifica dell'*ambiente di conoscenza* per questo *connettore*; ovviamente da ciò deriva che i *connettori* devono essere gli unici *operatori* con cui possono combinarsi gli *ambienti di conoscenza*.

Si utilizzano i *connettori* per conoscere e manipolare ciascun aspetto di ogni operatore; normalmente l'interazione tra un *operatore fruitore* e l'*ambiente di conoscenza* con cui si combina è effettuata per mezzo di più *connettori* in parallelo, ciascun *connettore* estrae l'informazione appropriata per le esigenze dell'*operatore fruitore* a cui è destinata; combinando i *connettori* in cascata si ha una operazione di estrazione gerarchica.

Ai *connettori* è collegato il concetto di *punto di vista* della classificazione: una medesima entità, in genere, è classificata secondo diversi criteri di classificazione.

Ecco, come esempio (*chi*, *come*, *dove*, *quando*, *perchè*), alcune delle informazioni di classificazione e l'utilizzo relativo:

- **identificazione** dell'*operatore* (**nome**).

  Contrariamente agli usuali linguaggi di programmazione, il **nome** di un operatore è un concetto totalmente differente dal ***simbolo*** che rappresenta l'*operatore*: la composizione tra operatori è una operazione di trasformazione di *simboli*, mentre il nome di un operatore è una entità di classificazione ed è un mezzo con cui l'utente interagisce con il singolo *operatore*; occorre notare che non è richiesto che il nome caratterizzi univocamente l'*operatore*, l'univocità è data dall'insieme dei criteri di classificazione che lo riguardano.

- **Scopo** dell'*operatore*: in genere questo criterio è rivolto all'utente finale, è usato per specificare ***cosa fa*** l'operatore ed è spesso il punto di partenza per *comporre un processo* servendosi di un'interfaccia per es. grafica.

- **Modalità di utilizzo** e **composizione delle funzioni**: sono i criteri di classificazione usati per specificare ***come fa*** un operatore a realizzare una funzione.

  Insieme all'informazione di *scopo*, questa potrebbe permettere la composizione automatica di operazioni per ottenere lo scopo desiderato.

- **Reperimento di informazioni** relative all'*operatore*: è il criterio collegato alla individuazione del supporto su cui l'*operatore* è rappresentato; è usato per specificare ***dove*** esso risiede, determina per esempio se esso è in memoria, se esso è accessibile per mezzo di operazioni di sistema (es.: caricamento di un programma da una libreria), se è reperibile presso un nodo IP, etc....

- **Operazioni temporali** relative all'*operatore*: questo criterio è collegato a tutte le operazioni che sono riferibili al tempo; usato per esempio per specificare ***quando*** attivare un *operatore* se questo è in combinazione con un *operatore* di esecuzione.

- **Priorità**: è ciò che determina l'**ordinamento**. È possibile applicare diversi criteri di priorità per rispondere alle esigenze di ordinamento dei vari insiemi di informazioni da gestire.

In alcune occasioni, per esempio nel caso in cui si interfacci AUCTORES per mezzo di un linguaggio algebrico, AUCTORES userebbe la *priorità* per determinare l'ordine di composizione degli operatori.

La funzionalità.

Come già affermato il sistema AUCTORES è interamente basato sulla composizione tra *operatori*; questa avviene o per la sostituzione del *connettore* con il risultato della composizione tra il *connettore* stesso e l'*ambiente di conoscenza*, o per la trasformazione del *simbolo* ad opera di un ***processore*** che costituisce il nucleo dell'*operatore* stesso.

Per ***processore*** si intende un meccanismo hardware o una entità software, esterna al sistema AUCTORES, che è attivato dal *simbolo* che lo rappresenta e che trasforma i *simboli* che riceve dall'*operatore fornitore* in altri *simboli* che saranno fruiti da *operatori* a monte della composizione. L'*operatore* che ingloba il *processore* è detto per ciò ***operatore semantico***.

Ciascuna composizione si effettua componendo l'*operatore semantico* con l'*operatore connettor*e e questo con l'*operatore fornitore*. Praticamente si considera come unico *operatore* la composizione tra l'*operatore semantico* ed i suoi *connettori*, mentre nella composizione con un *ambiente di conoscenza* si effettua la composizione tra il *connettore* e quest'ultimo, ed il risultato è fruito dall'*operatore semantico*.

Ciascun *processore* riceve informazioni da un determinato numero di ***canali***; ciascun *canale* consiste in un flusso di informazioni rappresentate in modo che sia elaborabile dal *processore* stesso ed ognuno di essi è "alimentato" dalla composizione tra il *connettore* ad esso collegato e l'*ambiente di conoscenza*: se per esempio il *processore* è una routine con un numero ***n*** di parametri, ciascun parametro è un *canale* del *processore* ed il tipo di informazione che esso elabora è molto probabilmente un indirizzo di memoria o una coppia indirizzo/lunghezza di un frammento di memoria.

Esiste un'equivalenza col λ calcolo, se, per esempio, **`P`** è un *processore* che ha 3 *canali*, esso equivale alla funzione:

`λx.λy.λz.(((P)x)y)z`

dove ognuno dei 3 *connettori* collegati ai *canali* corrisponde ad una delle 3 variabili *bound* **`x`**, **`y`**, e **`z`** della funzione **`P`** (gli operatori di interfaccia, ovvero i *connettori* svolgono in altri termini il ruolo di variabili) e l'attivazione del *processore* equivale alla sostituzione β (applicazione).

Dal punto di vista funzionale AUCTORES fornisce un insieme iniziale di *operatori* con le funzioni di:

- definire nuovi *ambienti di conoscenza* a partire da *ambienti di conoscenza* preesistenti; questi operatori utilizzano i *connettori* per strutturare l'*operatore risultato* secondo il loro annidamento.
- Interfacciare il sistema operativo ed il sistema di gestione delle interazioni utente per poter attuare uno scambio di informazioni con l'esterno in modo da poter creare nuovi *operatori* a partire da files e/o dall'interazione con l'utente oppure creare files e/o interazioni a partire dalle composizioni di *operatori*.
- Operare trasformazioni sugli ***operatori terminali***, cioè sugli *operatori* che mappano direttamente le caratteristiche dell'hardware come per esempio la memoria ed eventualmente i registri, il timer etc...
- Distruggere gli *ambienti di conoscenza* obsoleti conservando l'integrità referenziale di altri *ambienti di conoscenza* ad essi connessi.

Elaborazione parallela.

Se un *processore* è ben costruito, gli ***n*** *canali* che lo alimentano elaborano informazioni tra loro indipendenti e, poiché ad ogni *canale* è collegato un *connettore*, è possibile destinare l'elaborazione pertinente a ciascun *connettore* ad elaboratori differenti; l'elaborazione delle informazioni da parte del *processore*, core dell'*operatore*, avverrebbe al completamento delle ***n*** elaborazioni azionate dai *connettori*.

Inoltre poiché le informazioni provenienti dagli ***n*** canali del *processore* sono le **sole** informazioni necessarie al suo funzionamento, nel caso più generale, ovvero una diramazione ad albero delle composizioni, senza incroci, non si possono verificare situazioni di dead-lock durante le elaborazioni pertinenti ai *connettori*.

Si potrebbero avere situazioni di blocco soltanto nel caso in cui più *operatori* debbano comporsi con il medesimo *ambiente di conoscenza*, ma in questo caso la gestione è la stessa di quella in presenza di stati: stati differenti producono *operatori* differenti.

Possibile definizione di un'algebra di composizione.

Per il fatto che gli *operatori* di AUCTORES si compongono per fornire il risultato, cioè ad una coppia di *operatori* si fa corrispondere un *operatore risultato*, nel corso della realizzazione del progetto si vuole indagare sulla possibilità di strutturare la composizione di *operatori* in modo che si possano costruire ***strutture algebriche***.

Un'algebra che sia in grado di poter considerare "equivalenze aggiuntive" è in grado di operare sia semplificazioni in genere, sia adattamenti automatici e traduzioni basate sulle equivalenze. È facile intuirne le potenzialità; però il poter fondare un'algebra di composizione tra *operatori* non è banale, anzi comporta delle difficoltà.

La prima e maggiore difficoltà consiste nel fatto che il requisito fondamentale per poter costituire una struttura algebrica è la possibilità di composizione tra tutti gli elementi del sistema. Questo è un requisito che è fortemente condizionato dal comportamento dei *processori* che possono essere "alimentati" (quindi comporsi) solo da particolari strutture di dati; AUCTORES ha introdotto il concetto di *operatore*, composto dalle parti d'interfaccia oltre che dal *processore,* per poter ottemperare, in molte situazioni, a questo requisito.

Inoltre, ma questa è una difficoltà superabile con una adeguata organizzazione, in AUCTORES esistono parti d'interfaccia, i *connettori*, che sono essi stessi operatori; questo fa sì che si debbano considerare due strutture algebriche, quella relativa agli *operatori* "in toto" e quella relativa alle composizioni con i *connettori*; ovviamente esisteranno regole aggiuntive di composizione.

Altra notevole limitazione deriva dal fatto che i processori ammettono ***stati***, anzi ne sono condizionati: l'esistenza degli *stati*, derivanti in genere dalla composizione con operatori d'interfaccia col sistema ospite, impedisce, quando ciò accade, l'***associatività*** tra gli operatori.

AUCTORES, per poter, anche a fronte di queste difficoltà, comunque permettere la fondazione di un sistema algebrico, usa la classificazione per consentire agli operatori di valutazione delle equivalenze (quando e se ci saranno) di poter considerare composizioni di *operatori* se appartenenti o no a strutture algebriche e se ciò accade, a quali esse appartengano. In una interazione con l'utente, per esempio, si potrebbe considerare la sessione come formata da un numero finito di interazioni elementari dove in ciascuna di esse la composizione tra *operatori* è indipendente da stati e per cui valgono le equivalenze del sistema algebrico di cui fanno parte gli *operatori*.

Relativamente a molte operazioni, infatti, la composizione tra operatori potrebbe avere la *struttura* di ***gruppo***; questo porterebbe ad importanti conseguenze sulle equivalenze senza dover per forza applicare l'operazione stessa.

Per ciò, quando è possibile, si fa sì che gli *operatori* si comportino in modo che la composizione delle operazioni sia un *gruppo* e quindi valga per la composizione ogni proprietà dei *gruppi*:

- La ***chiusura*** è implicita: la legge che associa a due *simboli* il risultato della composizione produce un *simbolo*; il primo obiettivo di AUCTORES, che ha portato all'introduzione del concetto di *operatore*, è proprio questo.
- ***Associatività***: AUCTORES, quando possibile, si basa su questa proprietà nella composizione: per esempio i *connettori* si considerano parte dell'*operatore fruitore* e quindi si considera la composizione tra l'*operatore semantico* ed il *connettore*, ma quest'ultimo si compone con l'*ambiente di conoscenza* per fornire l'*operatore fornitore*; ciò determina di fatto l'equivalenza tra la composizione a sinistra e quella a destra del *connettore*.
- Per ogni operazione deve esistere un ***elemento neutro***, cioè tale che l'azione di esso sull'operazione lasci il risultato invariato. L'introduzione dell'*elemento neutro*, non sembra comportare eccessive difficoltà, ma la sua esistenza è ovviamente legata all'esistenza di un ***operatore inverso***.
- ***Operazione inversa***: Molte funzioni tipiche dei sistemi di programmazione non possono avere *operazione inversa* a meno di restrizioni che dovranno essere analizzate; esistono però comunque situazioni in cui si può avere un inverso sia in modo naturale che operando una forzatura.

Utilizzo di AUCTORES.

L'utente può utilizzare AUCTORES con le seguenti modalità:

- per mezzo di un **linguaggio formale**: un parser traduce i simboli nella struttura simbolica interna propria di AUCTORES: questo metodo è riservato a chi voglia sfruttare anche l'algebra di composizione. È ovviamente possibile avere diversi linguaggi formali, inizialmente è previsto un linguaggio essenziale dove i simboli siano tradotti direttamente nei simboli interni di AUCTORES.
- per mezzo di **strutture XML**: metodo per esportare ed importare operatori ed informazioni tra vari sistemi informativi.
- per mezzo di **compositori grafici**: è prevista la realizzazione di un software di composizione che contenga al proprio interno browser e sistemi di definizione per gli *ambienti di conoscenza*; poiché questo è il metodo di interazione più importante, ne viene fornita una descrizione più estesa col nome "CAD per la programmazione".
- per mezzo di **interfacce ad hoc**, per esempio per i disabili.

CAD per la programmazione.

È lo strumento offerto all'utente per la costruzione o modifica interattiva di *operatori*, attraverso un'interfaccia grafica.

L'applicazione grafica è costruita in modo da permettere l'esplorazione di tutte le informazioni contenute negli *ambienti di conoscenza* del sistema usando gli *operatori* di AUCTORES.

Il CAD per la programmazione mette a disposizione dell'utente un browser delle informazioni contenute nell'*ambiente di conoscenza* selezionato; per mezzo di esso l'utente può raggruppare gli *operatori* per "scopo", oppure per "nome" o per altre caratteristiche; una volta che l'utente abbia individuato gli *operatori* appropriati, questi possono essere composti per creare nuovi *operatori fruitori* o nuovi *ambienti di conoscenza*.

In particolare una delle informazioni di classificazione degli *operatori*, quella in base alla funzionalità, cioè "il cosa fa", è usato per poter fornire all'utente la visione di cosa sanno fare gli *operatori* a disposizione in modo da aiutare a comporre nuovi *operatori* per nuove funzionalità oppure per costruire, per l'*operatore fruitore* scelto (o appena costruito), l'*operatore fornitore* desiderato in modo da "chiudere i *connettori* ".

Questa locuzione è usata per indicare l'operazione di far corrispondere a ciascuno dei *connettori* dell'*operator*e in composizione le corrispondenti informazioni necessarie per l'elaborazione: quando tutti i *connettori* dell'*operatore* in questione sono stati "chiusi" ovvero già sostituiti dalle risposte corrispondenti che l'*ambiente di conoscenza* ha fornito, l'*operatore* è "eseguibile", cioè è diventato un *operatore terminale*, formato soltanto da parti eseguibili dall'hardware e può comporsi con l'*operatore esecutore* per fornire il risultato desiderato (visualizzando tutti gli operatori che compongono il risultato si ha, in genere, un albero).

L'*operatore* risultato, che ovviamente fa la sola cosa per cui è stato composto, può diventare un nuovo *operatore* se si sostituiscono dei rami già inseriti con un nuovo *connettore* detto ***connettore simbolico***; il nome deriva dal fatto che in questa occasione il *connettore* non è destinato a dare come risultato un "tipo", ma esso è usato per potersi combinare con un *ambiente di conoscenza* appositamente costruito per esso: lasciando aperto un *connettore simbolico* si ottiene un operatore generico che viene specializzato all'atto della composizione.

La sostituzione di *operatori terminali* (rami di albero) con *connettori simbolici* costituisce la programmazione **by example**, in cui si passa da un programma effettivamente funzionante, ad un *operatore* più generico.

Aspetto grafico.

L'aspetto grafico dell'interazione con il CAD di programmazione rispecchia i legami logici tra gli *operatori*: la parte semantica dell'*operatore*, ovvero la parte il cui cuore è il *processore*, è rappresentata da una icona che visualizza il "cosa fa" l'*operatore*; ciascuna porta del *processore* è rappresentata da un segmento che ha origine dall'icona; i *connettori* sono rappresentati da una figura geometrica che racchiude il nome del *connettore simbolico* con cui è terminato l'*operatore*: la figura che si ottiene è un grafo orientato (in genere un albero).

L'utente, dovendo comporre un *operatore fruitore* (da costruire o già pronto) con un *operatore fornitore*, ha gli strumenti per navigare contemporaneamente in due *ambienti di conoscenza*, uno da cui selezionare e comporre il *fruitore*, uno da cui produrre il *fornitore*.

La composizione tra *operatori* si compie selezionando dall'opportuno *ambiente di conoscenza* l'*operatore* o l'insieme degli *operatori* che costituirà il nuovo *ambiente di conoscenza*; mettendo in comunicazione il *connettore* con l'*ambiente di conoscenza* creato, il *connettore* si combina con esso e, in combinazione a sua volta con la parte semantica, dà origine ad un nuovo *operatore fruitore* i cui *connettori* sono i *connettori* dell'*operatore fornitore*.

Ovviamente se l'*operatore fornitore*, in combinazione con i *connettori* del *fruitore* originario fornisce *operatori terminali*, l'*operatore* risultato è un *operatore terminale* e quindi direttamente eseguibile.

La navigazione tra gli *ambienti di conoscenza* per la costruzione di *operatori* è effettuata utilizzando le informazioni di classificazione contenute negli *operatori*: il **nome**, che in risposta ad un appropriato *connettore* "nome" fornisce una identificazione umana dell'*operatore*, lo **scopo**, che fornisce la descrizione, visualizzata dal browser del CAD per la programmazione, della destinazione d'uso ("cosa fa") dell'*operatore* ed altre informazioni che possono essere utili di cui si è già trattato alla voce "Connettori".

Effettuata la costruzione del nuovo *operatore*, se questo non deve essere soltanto una entità estemporanea, il CAD per la programmazione richiede all'utente di specificare le informazioni di classificazione che saranno utili per il suo riutilizzo.

È quindi compito dell'utente definire lo scopo di ogni *operatore* ed avere cura di inserire ogni *operatore* in un *ambiente di conoscenza* opportuno, per esempio in base allo scopo stesso: l'ambiente grafico del CAD consente allora di presentare gli operatori già esistenti secondo una struttura gerarchica definita dalla loro collocazione in *ambienti di conoscenza*.

Altri metodi per la creazioni di operatori.

Oltre all'utilizzo del CAD per la programmazione, AUCTORES fornisce i tools di sviluppo usati per costruire il sistema, cioè l'insieme dei programmi e delle librerie per poter costruire:

***Operatori* nativi del sistema**

L'utente con le necessarie conoscenze di programmazione può, usando gli stessi strumenti usati per costruire AUCTORES, costruire altri *operatori* ***nativi***, le cui funzionalità sono svolte più o meno direttamente dall'hardware, prevedendo per lo sviluppo di tali operatori l'uso di linguaggi a basso livello come il C o l'assembler.

***Operatori* da programmi importati**

Per poter riutilizzare del software già scritto, è previsto un insieme di strumenti atti ad aiutare l'utente a trasformare programmi o parti di programmi delle usuali applicazioni in *operatori* AUCTORES.

L'interazione con gli altri *operatori* avviene in questo caso considerando l'input/output come *canali* dell'*operatore semantico*.
Ovviamente si avranno delle limitazioni nell'usabilità in quanto questi programmi, oltre a non avere l'efficienza degli operatori nativi, risentono dalla "dipendenza" da frameworks esterni e da parti run-time di varia natura per l'interfacciamento con i servizi del sistema operativo.
Nella fase iniziale, non disponendo AUCTORES di una propria gestione, a funzioni di questo genere sono demandate l'espletamento di funzionalità di più alto livello come la multimedialià, la condivisione, l'interscambio di informazioni e l'esecuzione remota di funzioni.

***Operatori* d'interazione uomo-macchina e con i servizi del sistema operativo**

Ad essi è demandata la generazione delle interfacce (grafiche e non) e la gestione della fruizione da parte degli *operatori* dei servizi del sistema ospite, cioè la gestione del file system, della memoria, dei temporizzatori, dei semafori, dei gestori degli eventi e delle comunicazioni, etc...
È previsto che possano in seguito essere sviluppate interfacce per le interazione con persone disabili, per esempio per mezzo di strumenti software di sintesi e di riconoscimento vocale, di gestione di dispositivi speciali come le tastiere Braille e di strumenti di puntamento comandati dal movimento oculare, etc...

***Operatori* matematici**

Il nucleo iniziale di AUCTORES fornisce soltanto le funzioni matematiche di base; successivamente saranno fornite le funzioni più avanzate, destinate alla grafica ed al calcolo scientifico (operazioni matriciali, interpolazione, soluzioni di sistemi di equazioni non lineari, calcolo simbolico, sistemi di equazioni non lineari e di disequazioni, integrazione di equazioni differenziali, etc...).

Supporto a grafica, realtà virtuale e multimedialità

I principi costitutivi di AUCTORES hanno avuto origine proprio nel campo della grafica in cui essa rappresenta un particolare aspetto nella descrizione di entità ed in cui l'aspetto grafico è definito per mezzo di operazioni, come per esempio la descrizione di un poligono per mezzo dell'operazione di tracciatura delle linee e la campitura di un'area con un colore, la ripetizione di elementi con operazioni di simmetria, etc...; per esempio, sono usate funzioni trascendenti e polinomiali per descrivere elementi grafici quali porzioni di superficie e curve.
Anche in questo caso, soprattutto per ciò che riguarda la multimedialità, inizialmente AUCTORES si servirà di strumenti esterni che entrano nel sistema secondo quanto specificato nel lemma "Operatori da programmi importati".

Operatori Esterni: il *Launcher*.

AUCTORES fornisce come ***operatore esterno*** un "**Launcher**" cioè un programma direttamente interfacciato al sistema che ospita AUCTORES che permette di eseguire i programmi, cioè gli *operatori terminali* costruiti con AUCTORES, nello stesso modo in cui vengono lanciate le applicazioni nel sistema nativo.

Ciò è permesso dal fatto che nella composizione tra *operatori fruitori* ed *ambienti di conoscenza*, si ottengono, se tutti i *connettori* sono stati *chiusi*, degli *operatori* che non abbisognano di ulteriori informazioni: la composizione implica la rimozione degli *operatori* che non partecipano al processo e quindi anche delle informazioni di classificazione, a questo punto non più necessarie.

Il risultato di tali composizioni è un programma eseguibile analogo a programmi ottenuti con i tradizionali sistemi: stesura del codice sorgente in un linguaggio di programmazione, compilazione, link con librerie, accesso al run-time del sistema.

Gli obiettivi della ricerca.

Lo studio si propone il raggiungimento dei seguenti obiettivi di ricerca:

1. (**OR 1**) Costruzione di un sistema software (***nucleo*** di AUCTORES) che ha lo scopo di organizzare la conoscenza in modo che essa possa essere opportunamente manipolata; in questo obiettivo di ricerca è inclusa la definizione dell'architettura e delle funzionalità di base.
2. (**OR 2**) Costruzione di un sistema di programmazione *end-user*, basato su interfaccia grafica che consenta di comporre le operazioni e di gestire l'importazione e l'esportazione dei programmi (*CAD di programmazione*).
3. (**OR 3**) Definizione di strutture algebriche in base a cui definire una o più sintassi formale di descrizione, da usare per descrivere l'elaborazione (programma) e come aiuto al ragionamento.

Nucleo di AUCTORES (OR1).

Il risultato atteso per questo OR consiste in:

- il prototipo del nucleo,
- i manuali,
- la pubblicazione scientifica.

Prima della fase realizzativa il presente progetto prevede una fase di studio e di speculazione teorica il cui risultato sarà la completa definizione delle caratteristiche e della logica di funzionamento degli operatori primitivi, cioè dell'insieme che costituirà la base di avvio della realizzazione di un prototipo.

In questa fase il lavoro svolto produrrà i manuali di descrizione, sia rivolti all'utente, sia allo sviluppatore: la speculazione sulle caratteristiche degli operatori daranno origine ai manuali utente, mentre le tecniche realizzative produrranno il manuale per gli sviluppatori ed i manutentori.

Il prototipo che si intende costruire ha lo scopo di dimostrare la potenzialità dello strumento. Poiché uno degli scopi consiste nella valutazione dell'efficienza, già in questa fase è prevista la realizzazione del codice di parellelizzazione dei processi e si terrà in opportuna considerazione la gestione della memoria con tecniche di garbage collection (ed eventualmente di virtualizzazione della memoria) strettamente collegate alla logica di AUCTORES, non si intende cioè solamente affidarsi a metodologie e software già consolidati, ma esaminare fino a che punto è possibile sfruttare l'architettura e l'utilizzo degli *ambienti di conoscenza* per ottimizzare la gestione della memoria.

Si intende realizzare una parte consistente di *operatori* appartenenti ai livelli di conoscenza relativi al funzionamento dell'hardware (Funzionalità di livello 1 o ***FL1***) e relativi all'interfacciamento con i sistemi ospiti, (livello ***FL2***) cioè con funzionalità analoga a quella degli usuali strumenti di programmazione.

Non è invece parte di questo obiettivo di ricerca la realizzazione di strumenti di gestione del sapere che non sia quello riferito all'informatica.

Nel corso dell'espletamento di ***OR1*** si indagherà anche sulla possibilità di poter strutturare AUCTORES in modo che esso possa essere indipendente da sistemi operativi.

Si indagherà su quali debbano essere i requisiti affinché AUCTORES gestisca direttamente l'hardware, e quali debbano essere le condizioni tali per cui possa essere costruito un interpreter hardware che realizzi la logica desiderata.

Le attività previste per il conseguimento di ***OR1*** sono:

- manuale utente,

- caratteristiche funzionali,
- program logic,
- parser,
- gestione della memoria,
- import/export operatori,
- operatori primitivi,
- operatori ***FL2***.

CAD per la programmazione (OR2).

La realizzazione di questo strumento ha lo scopo non soltanto di illustrare la potenzialità di AUCTORES, ma soprattutto quello di acquisire esperienza nelle operazioni di realizzazione delle applicazioni: si desidera cioè indagare su quali debbano essere le funzionalità più adatte a facilitare la programmazione.

Per tale scopo quindi si intende realizzare un sistema prototipale che fornisca l'operatività descritta al lemma "Utilizzo di AUCTORES - CAD per la programmazione".

Non è stato scelto un sistema software con cui/su cui realizzare questa interfaccia; la scelta cadrà su un sistema evoluto (C#, MONO, ECLIPSE) oppure molto più probabilmente sullo sviluppo di tale interfaccia come sito internet/intranet basato su HTML e derivati. La preferenza è accordata a questa seconda scelta per molteplici motivi: indipendenza dalla piattaforma, indipendenza da un sistema di gestione dell'interazione, pluriutenza.

Le attività previste per il conseguimento di ***OR2*** sono:

- manuale utente,
- caratteristiche funzionali,
- program logic,
- realizzazione del prototipo.

Definizione di strutture algebriche (OR3).

Questa attività avrà inizio non appena saranno state definite le caratteristiche funzionali degli operatori di base.

I risultati attesi per OR3 sono:

- verificare la fattibilità di tale approccio,
- definire i limiti di applicazione,
- pubblicazione scientifica,
- regole di semplificazione da introdurre in AUCTORES.

Finalità del progetto.

Il presente progetto ha essenzialmente una finalità scientifica.

Anche se la realizzazione di AUCTORES potrebbe avere una notevole ricaduta sul mercato dei prodotti software, tuttavia per esso non si può avere in modo immediato un notevole ritorno economico.

Ciò deriva dal fatto che esso non è prodotto da una potenza commerciale in grado di imporlo come standard e dal fatto che esso è rivolto ad utenti che non sono abituati alla programmazione; questo è

uno dei motivi per cui già in origine è stato previsto che i prodotti realizzati con AUCTORES avrebbero dovuto essere del tutto indipendenti da esso e direttamente eseguibili come normali programmi.

In ragione di ciò, non prevedendo cioè un ritorno economico diretto, il nucleo di AUCTORES, che è la parte che realizza la funzionalità, non sarà venduto, ma offerto in libera distribuzione ed in modalità "open source"; invece la parte accessoria, come per esempio un compositore grafico avanzato o dei gestori degli *ambienti di conoscenza*, ed in genere tutto ciò che riguarda gli strumenti di manipolazione e di produttività potrà essere venduto.

Ovviamente l'obiettivo fondamentale dell'utilizzo di AUCTORES è quello da parte della comunità scientifica.

Un sottoinsieme delle ricadute economiche è dato dalla realizzazione di giochi: dopo la realizzazione dei prototipi potrebbero essere realizzate versioni di AUCTORES che potranno essere ospitate nelle "consoles" per videogiochi, in modo da offrire uno strumento per una facile realizzazione di applicazioni in tale ambito.

In una visione a più lungo temine, si può ipotizzarne l'utilizzo come base per una gestione "intelligente" del desktop, in cui è l'utente finale stesso, che, con interazioni semplici del tipo di drag&drop, compone l'elaborazione che gli interessa, senza dover far ricorso ad applicativi specifici.

Tempi di realizzazione e risorse impiegate nel progetto.

Data la finalità del progetto la realizzazione sarà improntata al contenimento dei costi di sviluppo, mentre non è ritenuta una necessità la velocità: si prevede un impegno limitato come risorse umane, ma prolungato nel tempo.

Questa condizione oltretutto è imposta dal carattere di ricerca: lo sforzo maggiore consiste nella speculazione teorica e quindi non è possibile affidare compiti diversi a più sviluppatori. Ciò sarà possibile quando saranno state definite le caratteristiche funzionali di base.

Conclusioni.

Le funzionalità che il nucleo fornisce sono:

- ***Programmazione in piccolo***: ovvero la realizzazione della sola funzionalità software che interessa senza dover costringere l'utente alla programmazione di interfacce.
- ***Flessibilità***: ovvero una sufficiente granularità che consenta, a seconda della funzione, sia lo sfruttamento delle caratteristiche hardware che la manipolazione di applicazioni vere e proprie.
- ***Componibilità***: nuove funzionalità sono ottenute a partire da funzioni preesistenti (funzioni primitive) o precedentemente codificate; ciascuna funzionalità è disponibile in modo svincolato da applicazioni specifiche.
- ***Descrizione operativa di oggetti***: AUCTORES ha il doppio compito di "descrivere" le entità trattate oltre ad elaborarle. Inoltre "descrive" ed elabora anche processi asincroni e paralleli.
- ***Gestione di meccanismi di classificazione***: AUCTORES permette di strutturare le descrizioni operative relative ad un'entità in modo da rispondere in modo specifico alle operazioni che ci si attende che vengano compiute sull'entità stessa.
- ***Indipendenza da strutture linguistiche***: AUCTORES è svincolato da strutture sintattiche preformate in modo da poter essere il substrato logico e funzionale di linguaggi di svariata forma e natura: sia di eventuali sistemi di programmazione con interfaccia grafica, sia, eventualmente, di sistemi con interfaccia derivante da linguaggi naturali.
- ***Elaborazione simbolica***: è lo scheletro di AUCTORES; essa si attua per mezzo dei *connettori* e fornisce per ciascuno le forme equivalenti definite nel relativo *ambiente di conoscenza*.
- ***Interpretazione diretta da parte dell'hardware***: le strutture concettuali su cui si basa AUCTORES sono traducibili nelle usuali architetture hardware senza dover fare ricorso a gravosi interpreti. Si intende esplorare l'adattabilità di questo strumento ai possibili processori esistenti: AUCTORES permetterà lo sfruttamento dell'hardware vettoriale ed a multiprocessori e terrà conto di operazioni destinate ad hardware con architetture differenti all'interno dello stesso processo, in modo da poter programmare sub-processori ad architettura differente.

In una visione semplificata si ha la situazione:

- l'elaborazione simbolica, cioè l'insieme dei meccanismi di base di traduzione, è sempre presente e viene replicata nelle operazioni che vengono create a meno che queste ultime non risultino essere indipendenti da essa; essa si attua nella composizione per mezzo dei *connettori*.
- Le funzioni vengono create in genere per poter cooperare alla elaborazione simbolica, si costruiscono quindi di preferenza funzioni di traduzione; queste poi verranno utilizzate per realizzare gli scopi richiesti.
- Vengono costruite funzioni ad hoc di inizializzazione dei meccanismi di elaborazione: queste funzioni sono realizzate per mezzo di programmi "tradizionali" che si appoggiano al sistema operativo che gestisce il macchinario. Questi programmi sono considerati *processori*.
- La "manovrabilità" dall'esterno delle funzionalità, cioè i programmi (*operatori terminali*) costruiti con AUCTORES, è ottenuta usando un apposito operatore d'interfaccia (*Launcher*) che permette di integrare tali operatori nel sistema ospite.

Per ciò che riguarda il *CAD di programmazione*:

- l'utente sceglie un *operatore* chiave da una lista che gli viene presentata, compone questo operatore con altri operatori fino ad ottenere un *operatore fruitore* che compie le trasformazioni desiderate.
- L'utente crea per l'*operatore fruitore* che ha costruito l'*ambiente di conoscenza* relativo ad esso estraendolo da altri *ambienti di conoscenza* o dall'*ambiente di conoscenza* universale per mezzo di operatori di estrazione, in modo analogo a quanto attualmente viene fatto nei DBMS.
- L'utente può a questo punto creare una applicazione, ospite del sistema, che è del tutto autonoma dal sistema AUCTORES.

Il terzo obiettivo della ricerca ha lo scopo di verificare la fattibilità della costruzione di un sistema algebrico.

Si intende realizzare un formalismo, interpretabile dall'hardware, per mezzo del quale sia possibile descrivere concetti ed elaborarli: si vuole sostituire ad un approccio puramente algoritmico uno che sia essenzialmente formale (equivalenze e leggi di sostituzioni) e che richieda processi algoritmici soltanto nella fase finale di valutazione, in modo analogo a quanto accaduto in matematica quando si è passati, per esempio nella trattazione di problemi legati alle proporzioni, dagli algoritmi tipo 3 semplice e composto, alla soluzione di equazioni di primo grado.

Per ciò che riguarda i campi d'utilizzo:

- ***Small computing***: consiste nel rendere la programmazione il più rapida possibile e tale da non richiedere, all'utente che ha l'esigenza di definire applicazioni, competenze specifiche di informatica. AUCTORES permette anche una programmazione *by example*: a partire da un caso specifico, è possibile generalizzare gli *operatori* già costruiti semplicemente variandone i *connettori*.
- ***Elaborazione distribuita***: è attuata usando la risposta dell'*ambiente di conoscenza* che fa corrispondere all'informazione che deve essere fruita la procedura di estrazione.
- ***Sistemi esperti ed Intelligenza artificiale***: la realizzazione di sistemi di questo tipo è fortemente facilitata dalla struttura degli *operatori*, sia per quel che riguarda la classificazione sia per le regole di composizione tra *operatori*; vi sono infatti strutture decisionali (ottenute per mezzo di definizioni) incluse nell'*ambiente di conoscenza* che rispondono in modo diverso a seconda del *connettore* con cui deve combinarsi; inoltre la struttura di programmazione risulta essere analoga a quella fornita dai linguaggi funzionali con cui sono stati realizzati molti sistemi di intelligenza artificiale.
- ***Integrazione di sistemi e workflow management***: in questo caso è considerato *processore* ciascuno dei sistemi da comporre; le informazioni di input-output che danno vita al sistema sono considerate come *simbolo* da tradurre. I *partecipanti* al *workflow* sono considerati come particolari *processori*.
- ***Grafica, realtà virtuale e giochi***: diversi principi costitutivi di AUCTORES derivano dallo studio di un sistema grafico, basato sul concetto che la grafica è una particolare rappresentazione di una entità già descritta sotto diversi aspetti (anche non grafici); la rappresentazione grafica è ottenuta per applicazione di un operatore di visualizzazione che si compone con gli *operatori* di descrizione dell'entità.
- ***Modellazione***: viene fornita una programmabilità facile ed immediata unita alla disponibilità delle funzioni matematiche utili allo scopo, tali da permettere una descrizione della realtà che si vuole simulare; l'uso degli *ambienti di conoscenza* e dei *connettori* permette di inglobare sub modelli come risposte a svariate situazioni che sono rappresentate da *connettori*.