Un'estensibile architettura *object-oriented* per la simulazione di processi manifatturieri

A. Grieco, S. Zacchino, L. Castelluzzo
Dipartimento di Ingegneria dell'Innovazione - Università degli Studi di Lecce
via Arnesano 73100 Lecce - telefono: 0832 297251 - e-mail: antonio.grieco@unile.it

Abstract

Viene presentato un approccio object-oriented alla modellazione ed un'architettura software che ne implementa i concetti. La formulazione di tale approccio ha avuto l'obiettivo di ridurre i tempi necessari per l'ideazione di modelli di simulazione grazie ad una maggiore semplicità rappresentativa. L'implementazione software trae tutti i benefici che il paradigma object-oriented mette a disposizione in termini di riusabilità del codice, modularità ed estensibilità. Sono presentate le ultime estensioni create per soddisfare le esigenze più comuni del progettista della simulazione.

1 Introduzione

La progettazione di complessi processi di produzione o l'ottimizzazione di processi già esistenti richiedono adeguati strumenti per la simulazione. I professionisti devono essere in grado di fornire informazioni accurate sul sistema oggetto di analisi in tempi ridotti. È indispensabile, quindi, che essi siano in grado di creare modelli di simulazione in modo rapido e preciso. Le simulazioni condotte devono fornire indicazioni sia per la valutazione delle prestazioni del sistema studiato, che per la validazione del modello sviluppato.

Talvolta i software di simulazione richiedono un approccio alla simulazione non conforme alle realtà da simulare. In tali casi è richiesto un notevole sforzo astrattivo per descrivere un sistema in maniera differente rispetto a come appare agli stessi operatori del settore. L'approccio *object-oriented*, come riportato in [1], ha invece dimostrato una semplicità rappresentativa non comune ed ha portato nel campo della simulazione tutti i vantaggi che già offriva nell'ambito dell'ingegneria del software.

In questa sede vengono presentati l'approccio alla modellazione e l'architettura software adottata dal progetto DEOS[1]. Tali proposte hanno l'obiettivo di soddisfare i suddetti requisiti. Vengono inoltre indicati gli ultimi strumenti creati per facilitare il compito di comporre i layout di simulazione e di acquisire i risultati della simulazione, insieme con gli sviluppi futuri attualmente pianificati.

[1]Discrete Event Object-oriented Simulator, si vedano [2], [3] e [4].

2 L'approccio alla modellazione

L'attuale versione di DEOS consente di modellare sistemi complessi conformemente ai tipici concetti della teoria della simulazione, in termini di *risorse*, *entità* ed *eventi*.

2.1 Le componenti di base della modellazione

Le risorse, rappresentanti le componenti attive del sistema studiato, mettono in atto un comportamento manipolando le entità. Quando necessario, le risorse possono scambiare delle entità secondo un ben preciso protocollo di comunicazione, simulando il passaggio di materiale lavorato.

Ad ogni risorsa sono associati degli *attributi* che possono rappresentare dei parametri iniziali o delle variabili intermedie della simulazione. Durante l'esecuzione delle simulazioni, le risorse possono dotare anche le entità di attributi che ne descrivano lo stato. In tal modo è ad esempio possibile associare alle entità delle indicazioni sullo stato di lavorazione di prodotti intermedi di un processo produttivo.

Gli aspetti probabilistici della simulazione vengono gestiti mediante le cosiddette *random variable* associate alle risorse. In questo contesto, una random variable è intesa come una sequenza di nu-

meri pseudo-casuali con distribuzione assegnata che viene associata alla risorsa dallo sviluppatore della risorsa stessa. A run-time, il progettista della simulazione può modificare la distribuzione di una random variable in base alle proprie necessità di modellazione.

È possibile raggruppare insieme più risorse all'interno di particolari risorse denominate *group*. Un group esibisce gli attributi, le random variable e tutte le altre caratteristiche delle risorse che contiene. Anche tra le entità è possibile stabilire relazioni di aggregazione, ma con alcune differenze rispetto alle risorse: non esistono entità group, ma ogni entità può contenerne delle altre e l'entità contenitore non esibisce gli attributi delle entità contenute.

L'evoluzione delle simulazioni avviene grazie ad una *timeline* che consente alle risorse di schedulare eventi in opportuni istanti della simulazione. Ogni risorsa può schedulare determinati tipi di evento. Ad esempio, una risorsa rappresentante una macchina potrebbe generare degli eventi di tipo: *InizioLavoro*, *FineLavoro*, *Guasto*, etc.

Un layout di simulazione, insieme con tutte le sue componenti, viene chiamato *environment*.

2.2 Il protocollo di comunicazione

Ogni risorsa può essere dotata di porte d'ingresso e di uscita per la comunicazione con altre risorse. Ogni porta è identificata da un numero intero ed è accompagnata da una descrizione. Il progettista della simulazione stabilisce delle connessioni tra porte di uscita e porte di ingresso di risorse. Il passaggio di entità tra le risorse avviene attraverso tali connessioni seguendo un protocollo di comunicazione che prevede tre chiamate a metodi, come mostrato in figura 1.

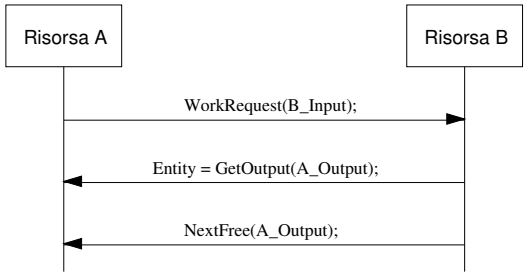

Figura 1: Sequenza di chiamate per il passaggio di entità fra risorse

Più in dettaglio: la risorsa *A* comunica alla risorsa *B* la volontà di inviarle un'entità chiamando il metodo *WorkRequest* di *B*. Quando *B* ritiene di poter acquisire l'entità oggetto della comunicazione, può farlo chiamando il metodo *GetOutput* di *A*. Infine, se *B* vuole sollecitare l'invio di una nuova entità può chiamare il metodo *NextFree* di *A*. Una variante a questo schema prevede che la comunicazione sia iniziata da *B* mediante la chiamata a *NextFree*, invece che da *A* con la chiamata a *WorkRequest*. Con questi schemi di chiamate è stato possibile modellare ogni esigenza di scambio di entità tra risorse fino ad ora necessaria.

Queste sequenze di chiamate, tuttavia, non esauriscono ogni necessità generica di comunicazione tra risorse. Talvolta, infatti, è necessario che una risorsa sia a conoscenza dello stato di un'altra, ad esempio monitorando i suoi attributi, prima di attuare determinati compiti. Per questo scopo possono essere utilizzati i metodi: *GetAttributeCopyFromInput* che consente di ottenere una copia di un attributo appartenente ad una risorsa il cui input è connesso ad un proprio output e *GetAttributeCopyFromOutput* per la stessa funzionalità in direzione opposta.

3 L'architettura software

I concetti sopra esposti hanno ispirato il progetto di un'architettura software per la simulazione a eventi discreti di processi di produzione.

3.1 Modularità, estensibilità e riuso del software

L'architettura software sviluppata dal progetto DEOS è stata progettata enfatizzando caratteristiche quali la modularità e l'estensibilità. Essa prevede, infatti, che l'applicazione sia costituita da un insieme di moduli eseguibili denominati file di *plugin* che comunicano attraverso interfacce standard. L'estensione delle funzionalità si realizza semplicemente aggiungendo tali moduli eseguibili scritti e compilati in C++, facendo uso di opportune librerie.

Questo tipo di architettura ha manifestato una buona manutenibilità. Inoltre ha permesso di diminuire al minimo lo sforzo di chi ha il compito di scrivere nuovi plugin, favorendo il riuso del codice grazie all'ereditarietà insita nel paradigma object-oriented. Infine, tale soluzione, non ha prodotto un impatto negativo sull'efficienza in termini di tempo di esecuzione.

3.2 L'interfaccia utente

Nello strato più alto dell'architettura si trova un editor visuale di environment. Il progettista che deve creare un layout di simulazione con DEOS, può usare questa semplice interfaccia a finestre che

gli consente di selezionare i plugin che gli occorrono creandone delle istanze in un environment. Un "doppio click" sulle icone rappresentanti le istanze dei plugin, permette di accedere ad un *inspector* che contiene tutte le proprietà accessibili all'utente. In tal modo è possibile, ad esempio, modificare gli attributi di una risorsa o la distribuzione delle random variable. La realizzazione delle connessioni tra risorse è ugualmente rapida e intuitiva. In figura 2 è riportato uno snapshot dell'editor visuale.

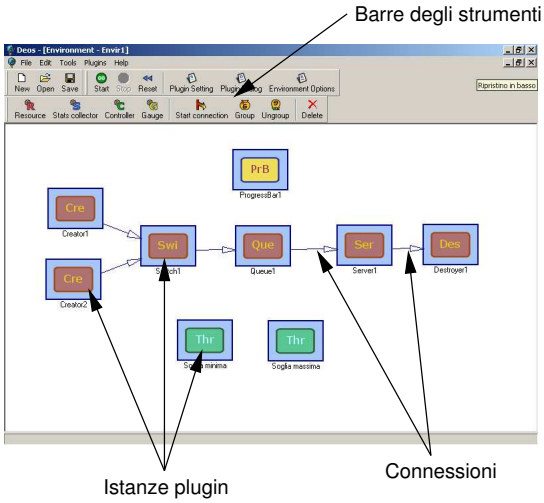

Figura 2: Snapshot dell'editor visuale

L'editor visuale è dotato anche delle tipiche funzionalità utili per migliorare la produttività personale, come il raggruppamento di più risorse in una singola risorsa di gruppo ed i comandi di tipo "*cut & paste*".

Attualmente l'editor visuale di DEOS è utilizzabile solo su una piattaforma proprietaria. Tuttavia l'architettura software è stata sviluppata tenendo conto di futuri *porting* su piattaforme differenti. A tale scopo, è stata creata una separazione tra il codice puramente "logico" e portabile, e quello legato alla piattaforma utilizzata che tipicamente cura gli aspetti relativi all'interfaccia grafica. Infatti, è stato piuttosto agevole portare il motore di simulazione su una piattaforma open source. Un porting dell'editor visuale richiederà invece uno sforzo maggiore e verrà realizzato in futuro.

3.3 L'interoperabilità

Un'altra caratteristica notevole dell'architettura di DEOS è la potenziale interoperabilità con altre applicazioni. Gli environment, insieme con tutte le istanze dei plugin in essi contenute, sono in grado di fornire una descrizione esaustiva del proprio stato nel linguaggio XML. Tale capacità permette ad altre applicazioni di accedere ai fattori ed ai risultati di una simulazione. Ad esempio, degli strumenti di *simulation optimization* potranno usare questa caratteristica per generare layout specifici da simulare e per leggerne i risultati.

Un'altra forma di interoperabilità potrà essere ottenuta sfruttando l'estensibilità dell'architettura, ossia creando dei plugin che fungono da interfaccia verso altre applicazioni durante l'esecuzione stessa della simulazione. Tali applicazioni partner potrebbero essere altri simulatori specifici che producono risultati utili in un environment DEOS o che importano dati da DEOS.

4 Strumenti utili per le simulazioni

I plugin utilizzabili da DEOS non rappresentano solo nuove risorse per simulazioni specifiche, ma possono anche essere utilizzati per aggiungere ulteriori strumenti di supporto alla simulazione. Tali strumenti possono essere: dei raccoglitori di statistiche, degli indicatori grafici per monitorare le variabili della simulazione durante l'esecuzione, dei *controller* per dotare le simulazioni di meccanismi di retroazione oppure degli environment di simulazione con determinate proprietà.

4.1 La raccolta di statistiche

La scelta delle informazioni da raccogliere durante l'esecuzione della simulazione non può essere a totale carico degli sviluppatori di risorse. Essi, infatti, non possono prevedere quali saranno tutti i dati che il progettista della simulazione vorrà monitorare. Inoltre un approccio del genere porterebbe gli sviluppatori a ricreare l'interfaccia utente per l'impostazione a run-time della raccolta di statistiche, per ogni nuovo plugin.

Per tali motivi si è deciso di ideare nuovi tipi di plugin appositamente pensati per la raccolta di informazioni durante l'esecuzione delle simulazioni. Questi strumenti sono stati denominati *data acquirer*. In base all'uso che un data acquirer fa dei dati raccolti, è possibile distinguerne vari tipi, come mostrato in figura 3.

Una prima distinzione si può effettuare tra i *gauge* e i *monitor*. I primi sfruttano i dati raccolti per produrre delle animazioni grafiche che consentono all'utente di osservare come si è evolve un sistema simulato durante l'esecuzione della simulazione stessa. I secondi invece non producono un risultato grafico immediato e possono essere ulteriormente sud-

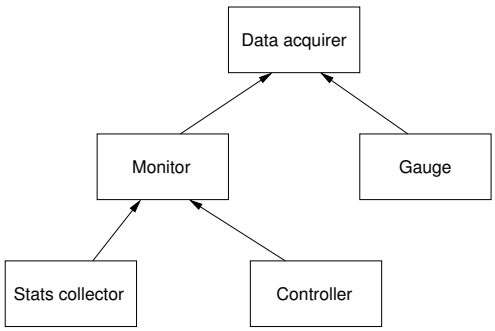

Figura 3: Gerarchia dei data acquirer

divisi in *stats collector* e *controller*. Gli stats collector hanno come scopo quello di catalogare i dati raccolti o di utilizzarli per effettuare calcoli statistici come media, varianza, etc. Infine, i controller introducono un meccanismo di retroazione negli environment in cui vengono impiegati, in quanto ad ogni osservazione dei dati verificano una determinata condizione ed in base ad essa modificano opportunamente degli specifici attributi di risorse.

4.2 Dettagli sui data acquirer

Si è tentato di trovare una modalità sufficientemente generale per specificare quali sono le informazioni a cui un data acquirer deve accedere. In tal modo si è cercato di mantenere un'unica interfaccia utente per configurare ogni tipo di data acquirer.

Il risultato della suddetta ricerca è dato dalle definizioni di seguito riportate.

- Ogni data acquirer è dotato di *input* da cui acquisisce i dati. Tali input possono essere gli attributi delle risorse, oppure i valori calcolati da altri data acquirer. Il numero di input ed il loro significato sono stabiliti in base allo scopo a cui il data acquirer è destinato.

- Ad ogni input è associato un *warner* che rappresenta la causa attivatrice delle osservazioni del corrispondente input. Un warner può essere la modifica di un attributo, l'esecuzione di un evento, l'azione di un altro data acquirer oppure la fine di un'iterazione della simulazione.

- In alternativa un data acquirer può essere dotato di *trigger* per la cattura delle informazioni. Un trigger è costituito dal warner che si è interessati a catturare e da un valore ad esso associato. Questo valore viene interpretato in modo diverso a seconda del motivo per cui il trigger viene catturato.

Un esempio di stats collector, dove i trigger possono risultare utili, è un contatore. Per configurare esattamente un contatore è sufficiente indicare gli warner che attivano i conteggi e, per ogni warner, il corrispondente valore da sommare al contatore.

4.3 I servizi dell'environment

Oltre ad avere il compito di contenere le istanze dei plugin utilizzate in un layout di simulazione, un enviroment offre una serie di servizi utili sia all'utente che ai plugin stessi. Tra l'altro, come già detto, l'environment stesso è un plugin.

Il compito di raccogliere le descrizioni XML di tutti i plugin di un layout è affidato agli environment. Ovviamente anche il ripristino del layout attraverso la lettura di informazioni XML è affidato all'environment. Esso raccoglie tali informazioni avendo cura di rispettare eventuali priorità affinché sia più agevole il ripristino.

Come è noto, ogni risorsa è in grado di generare delle sequenze di numeri pseudo-casuali con distribuzione assegnata. In realtà, le risorse non producono tali sequenze in proprio, ma si limitano a sfruttare una sequenza di numeri pseudo-casuali con distribuzione uniforme in $[0,\ 1]$ che trasformano opportunamente. La generazione di tali sequenze uniformemente distribuite e con seed assegnato dall'utente è a carico dell'environment.

Altri servizi offerti da un environment sono:

- fornire un link alla timeline;

- gestire le categorie di plugin accettabili;

- permettere più iterazioni di una simulazione.

5 Uso di script in DEOS

L'architettura software proposta è stata scritta in C++ e nello stesso linguaggio devono essere scritti i plugin che DEOS può utilizzare. Tale linguaggio ha permesso lo sviluppo di un prodotto software pienamente object-oriented che è in grado di raggiungere livelli di prestazione elevati. Tuttavia, l'utilizzo esclusivo di questo linguaggio richiederebbe uno sforzo notevole per la creazione di nuovi plugin ai progettisti della simulazione che non sono in grado di usarlo. Questo limite è stato risolto consentendo all'utente di configurare una risorsa mediante il linguaggio Tcl[2].

[2]Linguaggio di scripting open source ampiamente documentato in `www.tcl.tk`

5.1 Una risorsa configurabile con Tcl

Sfruttando la modularità dell'architettura, è stato creato un plugin risorsa completamente configurabile tramite uno script in linguaggio Tcl. In altre parole, quando tale risorsa viene inserita nell'environment si trova in uno stato "larvale", ossia non presenta all'esterno nessuna caratteristica e non mette in atto nessun comportamento durante lo svolgimento delle simulazioni. Tuttavia, attraverso un opportuno script Tcl è possibile dotare la risorsa sia di forma che di comportamenti, dove per forma viene inteso l'insieme degli attributi, delle random variable, dei tipi di evento, degli ingressi e delle uscite della risorsa.

Si noti che l'associazione tra la risorsa configurabile e lo script che la configura avviene a run-time, ossia mentre l'utente sta componendo il layout della simulazione e non all'atto della compilazione del plugin risorsa generica. Ciò consente di evitare del tutto la necessità di utilizzare un compilatore C++ per la creazione della risorsa.

In tabella 1 è riportato a titolo di esempio un frammento di codice C++ facente parte della logica di una risorsa ed il corrispondente in Tcl. Le quattro istruzioni C++ eseguono i seguenti compiti:

1. crea un'entità e la inserisce nel buffer d'uscita;
2. incrementa di uno l'attributo *Created Entities*;
3. invoca il metodo *WorkRequest* della risorsa connessa all'output uno;
4. valuta un attributo e se è vero chiama il proprio metodo *WorkRequest*.

5.2 Vantaggi e svantaggi dell'uso di Tcl

Altri software di simulazione permettono di comporre i layout di simulazione attraverso linguaggi di scripting come Tcl, ma non sempre consentono, come DEOS, di modificare gli stessi oggetti che prendono parte alla simulazione.

I vantaggi principali introdotti da questo plugin risiedono nella possibilità di utilizzare un linguaggio non compilato, ma interpretato, più semplice del C++ e a sua volta estensibile. Il principale svantaggio del plugin risiede, invece, nell'uso del motore per l'interpretazione degli script che provoca una notevole perdita di prestazioni nell'esecuzione delle simulazioni. Talvolta si è riscontrato un divario nei tempi di esecuzione di simulazioni simili, ma implementate con i due diversi linguaggi, di ben tre ordini di grandezza.

L'adozione dell'alternativa Tcl per la creazione di nuove risorse può essere utile nei casi in cui il tempo di esecuzione della simulazione non è un fattore critico ed è necessario diminuire i tempi di sviluppo del modello software. Tale alternativa risulta inoltre utile, quando non è possibile utilizzare un compilatore C++.

6 Obiettivi futuri

Attualmente il progetto DEOS ha rivolto la propria attenzione verso diversi obiettivi. Innanzitutto si vuole dotare DEOS degli strumenti necessari per gestire quantità fuzzy. In particolare si vuole estendere l'attuale timeline in modo da renderla in grado di ordinare degli eventi schedulati in istanti fuzzy secondo l'approccio indicato in [5]. L'utilizzo delle quantità fuzzy consentirà di svolgere delle simulazioni in cui è predominante il fattore incertezza, ossia in cui non sono note le distribuzioni statistiche delle variabili in gioco.

È in corso di valutazione la possibilità di utilizzare il linguaggio XSLT per definire i comportamenti delle risorse. L'adozione del linguaggio XSLT appare una naturale estensione del fatto che tutti i plugin sono in grado di descrivere il proprio stato in XML. Tale linguaggio, infatti, opera delle trasformazioni su dati espressi in formato XML generando nuovi dati in formato XML.

Infine, è in progetto la creazione di plugin in grado di interfacciarsi con l'esterno dell'applicazione. Questa estensione appare un passaggio obbligato per la creazione di un ambiente di simulazione distribuito a cui DEOS potrebbe partecipare o di cui potrebbe rappresentare il gestore.

Riferimenti bibliografici

[1] Narayanan, A. et al., "Research in Object-Oriented Manufacturing Simulations: An Assessment of the State of the Art." *IIE Transactions*, 1998

[2] Anglani, A. et al., "DEOS - A Discrete Event Object-oriented Simulation Environment." Conferenza annuale della Italian Society for Computer Simulation, Lecce, Italy, Dicembre 2000

[3] Anglani, A. et al., "DEOS: A new object-oriented simulation Environment. A validation study." Conferenza annuale della Italian Society for Computer Simulation, Lecce, Italy, Dicembre 2000

[4] Caricato, P., et al., "An open-source visual environment for discrete event simulation: DEOS."

C++	Tcl
`OutputEntities[0] = Environment->GetNewEntity(ThisTime, MyPlugin->GetId(), "");`	`addEntity Temp` `Entity Temp toOutput 0`
`*CreatedEntities = (long) *CreatedEntities + 1L;`	`incrAttr "Created Entities"`
`if (Outputs[0]->Partner) Outputs[0]->Partner-> WorkRequest(Outputs[0]->PartnerIndex);`	`WorkRequest external 0`
`if (!JustOnNextFree->GetChoice()) WorkRequest(0);`	`if [expr [getAttr "Create just on next free" getChoice] == 0] then { WorkRequest 0 }`

Tabella 1: Confronto tra C++ e Tcl

Conferenza annuale della Italian Society for Computer Simulation, Napoli, Italy, Dicembre 2001

[5] Nucci, F., et al., "Representation and use of uncertainty in discrete event simulation models" 10th European Simulation Symposium And Exhibition - Simulation in Industry, Notthingham, UK, Ottobre 1998