GENESIM: MODELLAZIONE E GENERAZIONE DI CODICE PER SIMULATORI DI SISTEMI DINAMICI

Giovanni A. Cignoni[1], Simone Masoni[2]
Dipartimento di Informatica,
Università di Pisa[1]
Dipartimento di Sistemi e Informatica,
Università degli Studi di Firenze[2]

La simulazione è uno strumento largamente utilizzato per sperimentare soluzioni diverse e garantire prestazioni e affidabilità dei sistemi ingegneristici. Il ricorso alla simulazione tramite modelli software ha il vantaggio dei bassi costi di esercizio del modello, ma realizzazione e validazione del software di simulazione possono essere comunque costose. L'articolo presenta GeneSim, un progetto open source per la specifica di sistemi dinamici in SysML e la generazione di codice a oggetti per la loro simulazione.

1. Introduzione

La domanda di sistemi con elevate prestazioni e bassi costi di sviluppo ed esercizio è in continua crescita in tutte le applicazioni di ingegneria, dalla meccanica alla componentistica elettronica. Ai processi industriali è chiesto di migliorare costantemente l'affidabilità, la flessibilità e la sicurezza dei sistemi. Caratteristiche che si costruiscono sperimentando soluzioni diverse e si garantiscono con verifiche e validazioni in uso dei sistemi. Queste esigenze, unite a considerazioni economiche, rendono sempre più significativa l'importanza della simulazione come strumento di valutazione e di analisi nei processi di progettazione e realizzazione di sistemi ingegneristici [01].

Il punto di partenza di un processo di simulazione è la creazione di un modello rigoroso e attendibile del sistema da studiare. L'utilizzo di modelli formali è la soluzione percorsa per eliminare i costi legati alla realizzazione e all'esercizio dei modelli fisici. Il successo di questo approccio è dovuto anche alla disponibilità di tecnologie informatiche che offrono linguaggi per la descrizione dei sistemi e strumenti per la realizzazione di modelli operazionali eseguibili – di software cioè – con i quali effettuare le simulazioni.

I principali vantaggi sono i bassissimi costi di esercizio del modello operazionale durante la simulazione e le accresciute possibilità di superare vincoli fisici, ad esempio comprimendo o dilatando i tempi di simulazione per velocizzare le attività o per osservare meglio i comportamenti del sistema. Tuttavia, i costi della realizzazione e della validazione del software di simulazione non sono trascurabili, specialmente quando avvengono utilizzando processi e strumenti di sviluppo tradizionali.

L'articolo presenta GeneSim, un ambiente open source per la generazione di codice per simulatori di sistemi dinamici [02]. Come progetto open source, GeneSim è costruito facendo riferimento a tecnologie aperte: come linguaggio di

specifica dei sistemi è adottato SysML, i formati di specifica in ingresso al generatore sono in XML, la generazione del codice è in gran parte ottenuta con trasformazioni XSLT ed è basata su librerie open source.

In sezione 2 sono descritte le componenti di GeneSim, le loro funzionalità e le loro caratteristiche principali. Sezione 3 è dedicata alla presentazione del modello di specifica dei sistemi. In sezione 4 sono brevemente discusse le soluzioni e le tecnologie adottate per la generazione del codice. Sezione 5 illustra in un semplice caso di studio la notazione SysML e le sue relazioni con il modello di simulazione. La sezione 6 è dedicata al confronto con la letteratura e con altri prodotti, commerciali e non, e agli sviluppi futuri del progetto.

2. L'ambiente GeneSim

GeneSim, per dirlo nel modo più conciso possibile, è un generatore di codice per sistemi dinamici. In una visione più ampia che comprende anche l'uso principale per il quale è pensato, GeneSim può essere definito come un ambiente per lo sviluppo di simulatori di sistemi dinamici basato sulla specifica ad alto livello dei sistemi e la generazione del codice dei loro simulatori. Come ambiente, GeneSim è composto da:

- un modello di specifica per sistemi dinamici che come notazione usa SysML secondo convenzioni definite;
- uno strumento di disegno per modellare in SysML i sistemi e ottenere loro specifiche in un formato XML definito;
- un generatore di codice che, dalle specifiche XML, produce librerie di classi in linguaggi a oggetti come C++, C#, o Java;
- un supporto a runtime per compilare il codice generato in librerie dinamiche con interfaccia standard e con supporto alle funzionalità di simulazione;
- uno strumento di prova per eseguire le simulazioni caricando dinamicamente le librerie generate e configurandone i parametri di simulazione.

UML [03] è nato per la modellazione di sistemi software, ma è stato usato anche per la modellazione di sistemi hardware e di processi aziendali. Da queste esperienze sono nate diverse proposte di dialetti ed estensioni di UML per l'uso in domini di modellazione specifici. La modellazione di sistemi ingegneristici è obiettivo del progetto *SysML* [04] promosso da un consorzio di organizzazioni interessate alla produzione di hardware di vario genere (NASA, Boeing, Motorola, IBM, tanto per citarne alcune). Le specifiche di SysML sono pubbliche e a fine 2005 il linguaggio [05] è stato sottomesso all'*Object Management Group* [06] per essere ufficializzato come standard.

SysML è un'estensione di un sottoinsieme di UML 2.0. Nella terminologia di UML, SysML è uno "strict profile". Il sottoinsieme di UML di partenza è stato volutamente ristretto per garantire al linguaggio una semantica definita. È quindi possibile definire in SysML modelli eseguibili, una caratteristica che, negli obiettivi del consorzio, è stata voluta proprio per far fronte alle esigenze di simulazione di sistemi. Per GeneSim questo è stato un motivo determinante per scegliere SysML come linguaggio di specifica di riferimento.

La specifica SysML di un sistema è codificata nel formato XML di GeneSim ed è usata per generare una libreria di classi in un linguaggio a oggetti. La libreria, insieme al supporto a runtime, fornisce funzionalità per:

- caricare, da un file di configurazione XML, lo stato iniziale del sistema;
- gestire simulazioni batch o continue (con tempi dilatati o ristretti) o real-time;
- salvare e ripristinare lo stato della simulazione;
- registrare su file i dati di simulazione.

La libreria di classi può essere integrata a livello sorgente in un'applicazione prodotta dall'utente, oppure compilata insieme al framework di runtime di GeneSim per ottenere una libreria dinamica con un'interfaccia standard per il controllo della simulazione. La libreria dinamica può essere caricata da un'applicazione prodotta dall'utente oppure dallo strumento di prova compreso nell'ambiente che permette di accedere a tutte le funzionalità messe a disposizione dal supporto a runtime.

3. Il modello per la specifica dei sistemi

Il modello di specifica adottato da GeneSim è stato definito sulla base della sua convenienza a fronte di un particolare dominio applicativo: la simulazione di sistemi fisici il cui comportamento è specificabile come risultato dell'interazione di più sottosistemi, deterministici e dipendenti dal tempo. La simulazione è continua, con il tempo discretizzato. Nel modello valgono le seguenti definizioni:

- *sistema*: un entità con un'interfaccia, uno stato e un comportamento; un sistema può essere semplice o composto;
- *sistema semplice*: un sistema che è definito solo in termini di se stesso;
- *sistema composto*: un sistema che è definito da un insieme di sottosistemi interagenti; i sottosistemi possono essere semplici o composti;
- *interfaccia*: il mezzo attraverso il quale un sistema interagisce con altri sistemi offrendo o richiedendo valori; l'interfaccia è composta dall'insieme degli attributi esterni ed esportati;
- *stato*: per un sistema semplice è l'insieme dei valori di tutti i suoi attributi; per un sistema composto è la composizione degli stati di tutti i suoi sottosistemi;
- *comportamento*: come lo stato di un sistema cambia in dipendenza del suo stato corrente e di eventi esterni; il comportamento di un sistema semplice è definito dalle sue funzioni; il comportamento di un sistema composto è definito dall'interazione dei suoi sottosistemi.

Stato e interfaccia sono specificati in termini di attributi, cioè di contenitori o di riferimenti per valori di tipi definiti:

- *attributo costante*: un contenitore per un valore che appartiene allo stato privato di un sistema semplice e che, dopo che il sistema semplice è stato inizializzato, non può cambiare;
- *attributo interno*: un contenitore per un valore variabile che appartiene allo stato privato di un sistema semplice; per ogni attributo interno esiste una *funzione* per aggiornarne il valore;

- *attributo esportato*: un contenitore per un valore variabile che appartiene allo stato pubblico di un sistema; per ogni attributo esportato di un sistema semplice esiste una *funzione* per aggiornarne il valore;
- *attributo esterno*: un riferimento a un valore variabile che un sistema richiede all'esterno;

Gli attributi, oltre al tipo e a un valore di inizializzazione, hanno vincoli opzionali che specificano l'intervallo ammissibile dei valori e l'eventuale unità di misura associata alla grandezza che essi rappresentano.

I valori correnti degli attributi definiscono lo stato corrente di un sistema semplice. Il comportamento di un sistema semplice è dato da una successione di aggiornamenti dello stato secondo funzioni specificate:

- *funzione*: definisce, in dipendenza dei suoi *parametri*, il valore nello stato aggiornato di un attributo interno o esportato;
- *parametro*: un riferimento al valore corrente di uno degli attributi di un sistema semplice o al valore di un parametro di simulazione;
- *parametro istantaneo*: un riferimento al valore già aggiornato di uno degli attributi interni o esportati di un sistema semplice;
- *parametro di simulazione*: un parametro predefinito, disponibile per tutti i sistemi e gestito dal contesto di simulazione.

I parametri istantanei sono previsti dal modello per rendere più efficiente l'aggiornamento dello stato di un sistema semplice, ad esempio fattorizzando parti comuni del calcolo delle funzioni. I parametri di simulazione rendono disponibili alle funzioni valori quali, ad esempio, il tempo trascorso, la durata del ciclo di simulazione corrente e il numero di cicli di simulazione eseguiti.

Nella specifica di sistemi composti, le interfacce dei sottosistemi componenti sono usate in parte per specificare l'interfaccia del sistema composto e in parte per specificare il suo comportamento come interazioni fra i sottosistemi:

- *collegamento*: una relazione di delega di interfaccia fra un sottosistema e il sistema composto che lo contiene; un collegamento delega un attributo esterno/esportato di un sottosistema come attributo esterno/esportato del sistema composto;
- *connessione*: una relazione di interazione fra le interfacce di due sottosistemi di un sistema composto; una connessione assegna a un attributo esterno il valore corrente di un attributo esportato.
- *connessione istantanea*: una connessione che assegna a un attributo esterno il valore già aggiornato di un attributo esportato.

Il modello prevede la verifica che nei collegamenti e nelle connessioni i vincoli di tipo, intervallo di valori e unità di misura siano rispettati: i tipi devono coincidere, l'intervallo di valori dell'attributo di partenza deve essere contenuto nell'intervallo di valori dell'attributo di arrivo, le unità di misura devono coincidere o avere una formula di conversione definita.

Le connessioni istantanee sono previste dal modello per avere la possibilità di mitigare i ritardi nella propagazione dei valori fra un sottosistema e l'altro. Combinazioni di connessioni istantanee e parametri istantanei delle funzioni di aggiornamento dello stato dei sistemi semplici possono dare luogo a cicli: il modello prevede la verifica dei cicli e, in assenza di cicli, la determinazione del corretto ordine di esecuzione delle funzioni di aggiornamento per rispettare i vincoli di istantaneità.

4. La tecnologia di generazione

Per scelta GeneSim genera codice. Rispetto all'alternativa di interpretare direttamente le specifiche di sistema, la soluzione scelta privilegia la flessibilità. Una libreria infatti è più facilmente integrabile con altre applicazioni, ad esempio quando il simulatore è usato come "stub" nella verifica di software di controllo di apparati hardware. Lo svantaggio della generazione di codice è la non immediata disponibilità del simulatore: fra la specifica del sistema e l'effettivo uso del simulatore l'utente deve svolgere attività di generazione, compilazione e integrazione del codice. Tuttavia questa limitazione è mitigata dalle caratteristiche del codice generato da GeneSim:

- il generatore, oltre alle classi, produce anche i file affinché il codice sia immediatamente e senza alcuna modifica compilabile come libreria dinamica;
- la libreria dinamica ha un'API standard e indipendente dal sistema simulato; l'API comprende un protocollo per descrivere l'interfaccia del sistema;
- l'ambiente comprende uno strumento di prova capace di caricare dinamicamente le librerie, richiedere la descrizione dell'interfaccia dello specifico sistema e costruirla per pilotare la simulazione.

In pratica, le attività di compilazione, lancio dello strumento di prova e caricamento della libreria dinamica sono automatizzabili e possono essere realizzate in serie al processo di generazione in modo trasparente all'utente. Il tempo necessario alla compilazione dipende dalla complessità del sistema e dalle opzioni di compilazione e può non essere trascurabile, ma è un costo sostenibile rispetto ai vantaggi in termini di efficienza e flessibilità del simulatore.

L'API standard della libreria dinamica è un ulteriore elemento di flessibilità: è possibile distribuire la libreria di simulazione senza il modello del sistema. Oltre alla praticità, a volte, può essere necessario che i modelli non siano visibili agli utenti del simulatore, per esempio per ragioni di sicurezza o per realizzare prove a scatola chiusa – data la natura open di GeneSim, si spera che l'opportunità di nascondere i sorgenti dei modelli non sia usata per motivi commerciali.

Il codice è generato secondo il processo schematizzato in fig. 1. La specifica SysML del sistema, codificata nel formato XML di GeneSim, è letta dal parser XML. Il parser, in accordo allo schema XML che definisce il formato, esegue la verifica sintattica, parte della verifica semantica e produce un *Document Object Model* (DOM). Su questo DOM sono eseguiti ulteriori controlli semantici da una componente del generatore realizzata ad hoc, in particolare:

- la verifica dei vincoli di tipo, intervalli e unità di misura nei collegamenti e nelle connessioni;

- la verifica della non esistenza di cicli determinati da parametri e connessioni istantanee.

Se i vincoli sono rispettati e non esistono cicli, la componente ad hoc modifica il DOM per ottenere la rappresentazione astratta del codice che implementa il simulatore. In particolare nel DOM modificato:

- sono state definite le istanze dei sottosistemi di cui è eventualmente composto il sistema in ingresso;
- è stato determinato il corretto ordine di esecuzione delle funzioni di aggiornamento per rispettare i vincoli di istantaneità;
- sono state introdotte le funzioni di conversione fra unità di misura;
- sono state aggiunte le necessarie chiamate al runtime di simulazione.

L'ultimo passo di generazione è eseguito applicando al DOM modificato un insieme di trasformazioni XSLT. Tramite queste trasformazioni sono ottenuti i diversi prodotti del generatore:

- la libreria di classi nel linguaggio di programmazione scelto;
- il progetto con le istruzioni di compilazione della libreria di simulazione;
- il file di configurazione di default.

L'adozione di XSLT in questa ultima fase accresce l'apertura del generatore. Dalla stessa rappresentazione astratta del DOM modificato è possibile generare codice per piattaforme diverse, per ambienti di sviluppo diversi e anche in linguaggi a oggetti diversi (per esempio, C++, Java o C#). L'estensione del generatore rispetto a questi aspetti comporta, in pratica, solamente la realizzazione dei necessari fogli di stile XSLT.

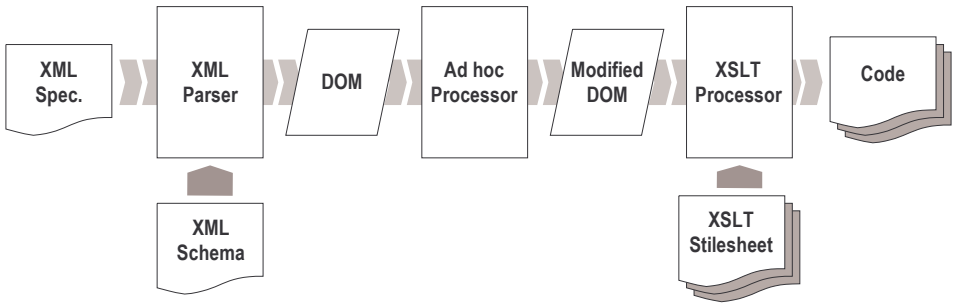

Figura 1. Il processo di generazione del codice in GeneSim

L'utente in generale non ha necessità di intervenire sul codice generato. L'unica parte di codice per la quale potrebbe esserci esigenza di intervento – più che altro per ragioni di praticità/efficienza rispetto ai tempi di generazione e compilazione – è quella corrispondente ai metodi che implementano le funzioni di aggiornamento dello stato. Il generatore confina questi metodi in file separati e chiaramente identificati.

Il file di configurazione, anch'esso in un formato XML definito, può essere utilizzato, al momento del caricamento della libreria dinamica, per definire lo stato iniziale di un sistema. Il file di configurazione di default non è necessario, viene

generato a beneficio dell'utente come modello per la definizione di file di configurazione specifici.

Altre opzioni di generazione riguardano l'inclusione, nel codice generato, delle chiamate per utilizzare le funzionalità del supporto a runtime, quali ad esempio, la lettura di file di configurazione, il salvataggio dello stato della simulazione, la registrazione dei dati di simulazione.

5. Un esempio di modellazione

In fig. 2 è mostrato un esempio di specifica: una resistenza elettrica modellata come sistema semplice. Il sistema semplice è specificato come un *block*: lo stato, l'interazione con l'esterno e il comportamento del sistema sono specificati, rispettivamente, con attributi, interfacce e operazioni.

Nell'esempio, i valori che determinano la resistenza (R) e l'intervallo di voltaggio ottimale (Vmin, Vmax) sono specificati come attributi costanti. Identificatore, tipo e valore iniziale sono definiti secondo la notazione SysML standard, la *property-string* {const} li definisce come attributi costanti. Informazioni aggiuntive su un attributo interno sono specificate come note (in fig. 2 ne sono mostrate "aperte" solo alcune). SysML prevede che, nelle note, i commenti siano distinti dalle informazioni che definiscono vincoli, che invece sono espressi fra graffe. Per esempio, per R, Ro e Vi, l'unità di misura è un vincolo.

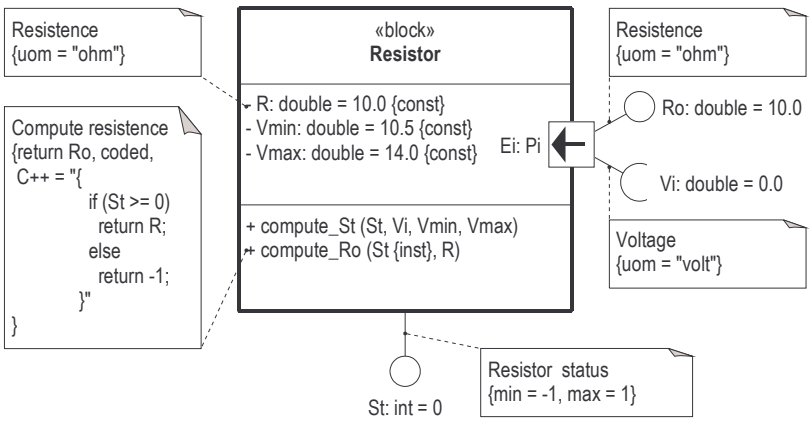

Figura 2. Una resistenza modellata in SysML

La resistenza ha due vie di interazione con l'esterno, una legata al comportamento elettrico e l'altra finalizzata al controllo dello stato della resistenza. La prima è specificata da una *flow port*, l'elemento introdotto da SysML per modellare lo scambio di grandezze fisiche. La porta è specificata da un'interfaccia offerta e un'interfaccia richiesta. La prima corrisponde nel modello GeneSim a un attributo esportato che mette a disposizione del resto del circuito il valore di resistenza (Ro). La seconda, corrispondente a un attributo esterno, chiede al circuito la differenza di potenziale presente ai morsetti della resistenza (Vi).

La porta ha un identificatore (Ei) e un tipo (Pi). La notazione è di fatto una definizione di tipo di porta: un sistema potrebbe avere altre porte di tipo Pi con lo stesso insieme di interfacce offerte e richieste. In SysML la notazione delle flow

port prevede l'indicazione della direzione del flusso. Nell'esempio la freccia entrante descrive il fatto che la resistenza, tramite la porta Ei, riceve potenza elettrica fornita da un generatore.

L'interazione di controllo è specificata direttamente con un'interfaccia offerta (St) che mette a disposizione lo stato della resistenza codificato da un intero con tre valori: accesa, spenta e bruciata. L'intervallo di valori possibili è specificato come vincolo nella nota. Se l'interazione di controllo fosse stata più complessa e avesse previsto più di un'interfaccia, le buone prassi SysML avrebbero richiesto una *service port*, l'elemento introdotto da SysML per modellare porte attraverso le quali si scambiano comandi e informazioni di stato.

Le porte aggiungono informazione utile all'interpretazione della specifica e alla verifica di vincoli. Sono utili anche per rendere la notazione più compatta e intelligibile. Sono però ininfluenti rispetto alla specifica del sistema che, ai fini dell'interazione con l'esterno, è data solamente dalla specifica delle interfacce.

Il comportamento dei sistemi semplici è definito da funzioni che specificano come i valori degli attributi interni e degli attributi esterni cambiano nel tempo in dipendenza dello stato del sistema, cioè dell'insieme dei valori di tutti gli attributi. Per ogni attributo esterno e per ogni attributo interno deve quindi essere definita una funzione di aggiornamento. Nell'esempio di fig. 2 sono definite funzioni per St e per Ro. Secondo la notazione SysML standard, di ogni funzione è specificato il *prototype*. Nella definizione dei parametri di una funzione è possibile chiedere, tramite la *property-string* {inst}, l'uso del valore aggiornato di un parametro invece di attendere il ciclo di simulazione successivo. Nella nota associata a una funzione è specificato il vincolo che la lega all'aggiornamento di un attributo e come la funzione viene calcolata. Nel caso di fig. 2 il corpo della funzione è definito direttamente come codice C/C++, ma è possibile riferire file esterni o definire le funzioni come tabelle e interpolare i valori intermedi.

In fig. 3 è rappresentata la modellazione di un semplice circuito composto da una batteria e una resistenza. Il sistema assemblato è ancora una definizione di *block*, ma è specificato come un insieme di sottosistemi interagenti, ognuno dei quali è una *part* ottenuta come istanza di un *block* precedentemente definito: B è un'istanza di LABattery e R è un'istanza del Resistor specificato in fig. 2.

Le interfacce dei sottosistemi sono usate per connettere i sistemi, come fra le flow port Eo di B ed Ei di R, oppure sono delegate per definire l'interfaccia del sistema assemblato, come nell'esempio accade per le interfacce utilizzate per il controllo dello stato dei due sottosistemi. Le connessioni possono essere specificate in una notazione compatta direttamente fra le porte. L'indicazione di tipo e la corrispondente definizione data nella specifica del sottosistema permettono di risolvere la connessione come insieme di connessioni fra interfacce (in fig. 3 mostrati come "ingrandimento"). La connessione dell'esempio è istantanea, proprietà descritta dalla presenza della property-string {inst}.

L'esempio di fig. 3 mostra anche un altro uso delle informazioni di configurazione. Con la property-string {conf} si assegna all'attributo costante R dell'istanza della resistenza un valore diverso da quello definito nella specifica in fig. 2.

Questa notazione permette di usare, in un sistema composto, più istanze dello stesso sottosistema con parametri diversi. Le informazioni di configurazione sono caricate dinamicamente senza bisogno di rigenerare il codice.

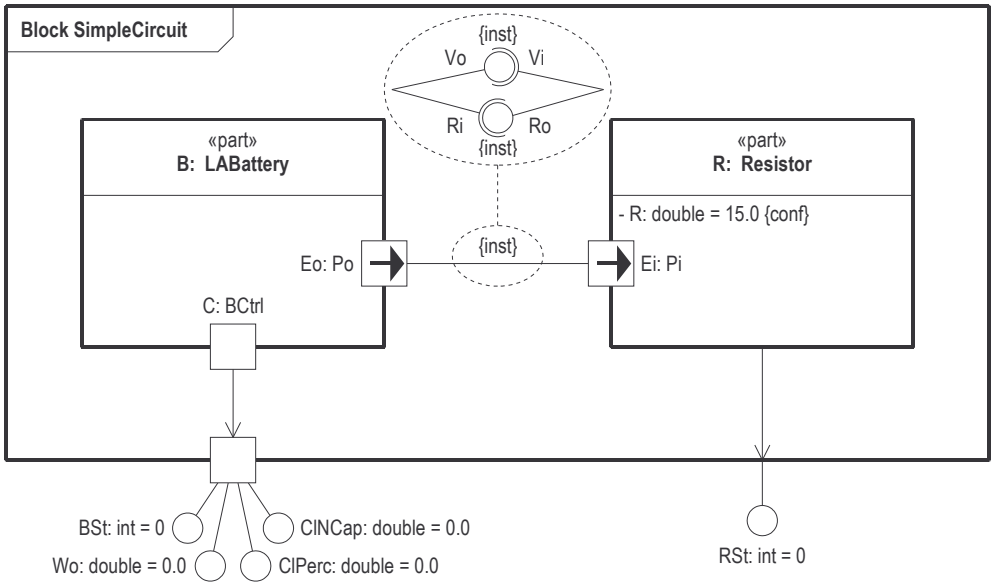

Figura 3. Un semplice circuito modellato in SysML

I sottosistemi presentati nell'esempio fanno parte di un caso di studio reale in cui GeneSim è stato usato per simulare il comportamento di scarica del complesso di batterie che forniscono alimentazione ai sistemi di un robot.

6. Confronto e sviluppi futuri

Linguaggi di specifica derivati da UML sono già stati proposti nel contesto della simulazione. *MILAN* [07], ad esempio, adotta UML per l'integrazione di modelli di simulazione diversi. La generazione di codice di simulazione da specifiche UML è affrontata anche in [08] per generare codice Java per la simulazione di sistemi discreti. La necessità di poter specificare i sistemi come composizione di sottosistemi definiti indipendentemente è sottolineata [10]. I vantaggi delle tecnologie XML-XSLT per la generazione di codice sono discussi in [11].

Esistono diversi prodotti e progetti affermati che offrono funzionalità sofisticate per la simulazione a partire da specifiche ad alto livello dei sistemi. Tuttavia spesso adottano linguaggi di specifica propri ai quali manca la forza di "linguaggio comune" che SysML vanta attraverso la sua stretta parentela con UML. Cadono in questa categoria progetti open source come *DSOL* [12] e *DEx* [13] e, sul fronte commerciale, pacchetti dedicati all'industria come *Dymola* [14] basato sul linguaggio *Modelica* [15], o prodotti molto diffusi come *Simulink* [16].

Un prototipo di GeneSim è stato sviluppato come tesi di laurea [17]. A oggi, il generatore legge specifiche XML e produce codice C++ direttamente compilabile per MS Windows come *Dynamic Link Library* (DLL) usando *Bloodshed DevC++* o *Code::Blocks*, due ambienti di sviluppo open source basati sul compilatore GNU GCC, oppure usando *MS Visual Studio*. Lo strumento di prova ca-

rica dinamicamente i DLL di simulazione e i loro file di configurazione, e permette di pilotare la simulazione interagendo con il sistema, di salvare lo stato della simulazione e di registrare su file i dati in formato *comma separated values* (CSV). Una nuova versione dello strumento di prova e uno strumento per il disegno delle specifiche in SysML sono in via di sviluppo.

Per assicurare la portabilità su piattaforme diverse da MS Windows, GeneSim è realizzato in C++ utilizzando componenti open source: *Xerces* [18] come parser XML e *Xalan* [19] come processore XSLT e, per gli strumenti grafici attualmente in via di sviluppo, *wxWidgets* [20].

Sul versante del modello di specifica gli sviluppi futuri prevedono l'adozione di altri diagrammi SysML quali *constraints* e *parametrics* per la definizione di vincoli da verificare durante la simulazione e per la specifica delle funzioni come espressioni indipendenti dal linguaggio scelto per la generazione del codice.

7. Riferimenti

[01] J.S. Carson, (2005), *Introduction to Modeling and Simulation*, atti della Winter Simulation Conference 2005.

[02] GeneSim Project, (acceduto a giugno 2006), http://genesim.sourceforge.net/.

[03] G. Booch et al., (2004) *The UML Reference Manual (II ed.)*, Addison-Wesley.

[04] SysML Project, (acceduto a giugno 2006), http://www.sysml.org/.

[05] SysML Partners, (2005), System Modeling Language Specification 1.0a.

[06] Object Management Group, (acceduto a giugno 2006), http://www.omg.org/.

[07] A. Ledeczi, et al., (2003), *Modeling Methodology for Integrated Simulation of Embedded Systems*, in ACM Transactions on Modeling and Computer Simulation, Vol. 13, No. 1.

[08] L.B. Arief, N.A. Speirs, (2000), *A UML Tool for an Automatic Generation of Simulation Programs*, atti del Second International Workshop on Software and Performance, ACM.

[09] C. McLean, S. Leong (2002), *Simulation standards: a framework for standard modular simulation*, atti della Winter Simulation Conference 2002.

[10] W. Harrison et al., (2000) *Mapping UML designs to Java*, atti della 15th Conference on Object-Oriented Programming, Systems, Languages, and Applications, ACM.

[11] G. Swint et al., (2005) *Clearwater: Extensible, Flexible, Modular Code Generation*, atti della 20th International Conference on Automated Software Engineering, IEEE.

[12] DSOL, (acceduto a giugno 2006), http://www.simulation.tudelft.nl/dsol/.

[13] DEx Simulation Framework, (acceduto a giugno 2006), http://dextk.org/dex/.

[14] Dymola, (acceduto a giugno 2006), http://www.dynasim.se/.

[15] Modelica, (acceduto a giugno 2006), http://www.modelica.org/.

[16] Simulink, (acceduto a giugno 2006), http://www.mathworks.com/products/simulink/.

[17] S. Masoni, (2005), *Un generatore di codice per simulatori di sistemi dinamici*, Tesi di Laurea in Informatica (rel. G.A. Cignoni), Università degli Studi di Firenze.

[18] Xerces, (acceduto a giugno 2006), http://xml.apache.org/xerces-c/.

[19] Xalan, (acceduto a giugno 2006), http://xml.apache.org/xalan-c/.

[20] wxWidgets, (acceduto a giugno 2006), http://www.wxwidgets.org/.