Statemate XMI exporter

Esportazione di modello SNCF da Statemate a un formato XMI

FORMAL METHODS && TOOLS GROUP
ISTI – CNR
Area della Ricerca CNR
Via G. Moruzzi,1
I-56124 - Pisa
Tel. 050.3152920 - Fax 050.3152810

Revisioni

Revisione	Data	Nome	Pagine	
	20 Febbraio 2004		48	Versione preliminare

Revisione	1
File Name	Documento_XMI_exporter.doc

	Nome	Firma	Data
Scritto da:	Michele Banci		20 Febbraio 2004
Approvato da:			

Indice

INDICE DELLE FIGURE

1 Introduzione

L'obiettivo di questo lavoro è l'esportazione di modelli di macchine a stato generati con il software Statemate Magnum e la trasformazione di tali modelli in un formato testuale ben formalizzato, lo XMI (XML Metadata Interchange).

Nel capitolo 2 viene discusso il funzionamento del programma I-Logix Statemate Magnum e, con particolare attenzione alla vista funzionale, vengono brevemente descritte le modalità di sviluppo di sistemi con tale software.

Nel capitolo 3 viene trattato in dettaglio il formato XMI, che è il formato obiettivo nella trasformazione dei modelli.

Nel capitolo 4 viene descritto il funzionamento del software xmi_exporter, partendo dalle strutture di memorizzazione utilizzate per prelevare le informazioni sui modelli da Statemate Magnum.

Nel capitolo 5 vengono forniti tre esempi di statechart e di activity-chart sui quali è stato testato il software xmi_exporter e viene riportato il listato XMI di uscita.

Nella parte successiva vengono presentate le istruzioni per l'utilizzo di xmi_exporter.

2 I-Logix Statemate Magnum

2.1 Generalità

I-Logix Statemate Magnum è il più completo tool di sviluppo per la modellazione grafica e la simulazione di embedded-system complessi. Statemate Magnum fornisce un collegamento diretto e formale fra i requisiti dell'utente e l'implementazione del software attraverso la creazione di una specifica completa ed eseguibile, che rappresenta con precisione e chiarezza le finalità del sistema da sviluppare. La specifica può essere eseguita, o simulata graficamente, in modo che il sistemista possa valutare se il comportamento e le interazioni fra i componenti del sistema sono corrette, realizzando inoltre un feedback con l'utente finale.

2.2 Struttura di un modello

La costruzione di un modello può essere considerata la trasformazione di idee e descrizioni informali in una descrizione concreta che utilizza concetti e terminologie predefinite. Nell'approccio di StateMagnum, la descrizione del modello è organizzata in tre viste, o proiezioni, del sistema: la funzionale, la comportamentale e la strutturale.

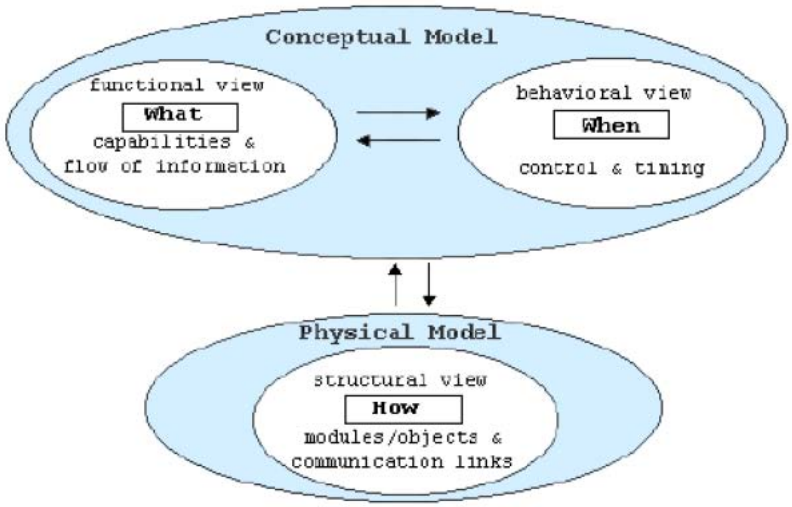

Figura 1 – Viste di un modello

La vista funzionale rappresenta il "*cosa*". Essa descrive la funzione del sistema, i processi e gli oggetti, chiamate anche attività, in modo da rappresentare le sue possibilità. Questa vista include anche gli ingressi e le uscite delle attività.

La vista comportamentale rappresenta il "*quando*". Essa descrive il comportamento del sistema nel tempo, la dinamica delle attività e il loro controllo.

La vista strutturale rappresenta il "*come*". Essa descrive i sottosistemi, i moduli o gli oggetti costituenti il sistema reale e le comunicazioni fra essi.

Le prime due viste costituiscono il modello concettuale del sistema, mentre la vista strutturale è considerata il modello fisico. La connessione principale fra il modello concettuale e quello fisico è data dai moduli della vista strutturale che sono responsabili dell'implementazione delle attività nella vista funzionale.

Le tre viste del modello di un sistema sono descritte, per l'approccio degli sviluppatori di Statemate Magnum, in tre linguaggi grafici diversi: activity-charts per la vista funzionale, state-charts per la vista comportamentale e module-charts per la vista strutturale. Informazioni non grafiche riferite alle viste stesse e alle loro interconnessioni sono memorizzate in un dizionario dei dati.

Le activity-charts possono essere viste come diagrammi multi-livello di flussi di dati. Le attività sono organizzate in gerarchie e connesse in base ai flussi di informazioni interscambiati fra loro.

2.3 Rappresentazione della vista funzionale di un modello

La vista funzionale di un modello viene costruita attraverso la tecnica di decomposizione, che può essere eseguita in due modi:

1. *decomposizione basata su funzioni*: le attività sono funzioni di sistema;
2. *decomposizione basata su oggetti*: le attività sono oggetti di sistema.

In entrambi i casi, il risultato consisterà nella activity-chart e in un Data Dictionary, che contiene informazioni addizionali riguardo agli elementi che appaiono nella activity-chart.

In una activity-chart le attività sono rappresentate da rettangoli e la relazione di sub-attività è rappresentata dall'incapsulazione di rettangoli. All'interno del rettangolo appare il nome dell'attività. Via via che l'incapsulazione aumenta di livello, le nuove attività vengono disegnate all'interno delle *parent activities*. Le attività che non hanno discendenti sono attività *basic*, le altre sono *non-basic*.

La descrizione funzionale di un sistema può consistere di activity-chart multiple collegate insieme, ognuna delle quali descrive una porzione del sistema.

Ogni activity-chart contiene una *top-level box*. Questo box rappresenta la top-level activity del grafico. I suoi bordi separano questa attività e la sua descrizione interna dal resto dell'ambiente.

Le activity-chart comunicano fra loro attraverso *flow-line*, che rappresentano gli input e gli output della activity-chart. Questi flussi di informazioni non trasmettono solo dati, ma possono trasmettere anche comandi e segnali di sincronizzazione. Le flow-line sono disegnate attraverso frecce etichettate. Ogni etichetta denota un singolo elemento di informazione che scorre lungo la linea (ad esempio, un dato, una condizione o un evento), ma può anche rappresentare un gruppo di questi elementi. Un gruppo di diversi elementi informativi è chiamato *information-flow*. E' possibile identificare flussi di dati e flussi di controllo. Inoltre una flow-line parte da una *source activity*, che produce l'elemento informativo, e arriva alla *target activity*, che prende in input tale elemento. Le flow-lines possono essere di due tipi: *data flow-line*, disegnate come frecce continue, e *control flow-line*, disegnate come frecce tratteggiate.

Una flow-line può essere vista come un tipo di unità di memoria, in quanto i dati prodotti dall'attività sorgente sono disponibili all'attività di destinazione anche quando l'attività sorgente non è più attiva. Quindi è spesso più naturale incorporare un esplicito *data-store* nel grafico, che rappresenta l'informazione memorizzata per essere utilizzata in seguito. I data-store sono rappresentati da rettangoli con in lati verticali tratteggiati, sono sempre basic e non possono contenere altri data-store o activity.

2.4 Accesso ai progetti

Il software Statemate Magnum pone l'utente nella condizione di dover utilizzare il proprio formalismo nella stesura dei progetti; sulla base di tale formalismo i progetti vengono salvati dal programma in una apposita struttura. Fortunatamente, a fianco di un sistema strutturato con un approccio di tipo closed, Statemate Magnum offre anche la possibilità di esportare gran parte delle informazioni salvate nei progetti: con il software viene infatti rilasciata una libreria (dataport) (con interfaccia in linguaggio C) che permette l'esportazione dei dati. Nella documentazione rilasciata vengono descritti i metodi di accesso al database dei progetti e di utilizzo delle funzioni implementate. Dette funzioni sono classificabili essenzialmente in tre tipologie distinte:

- ***Single Element Functions***: sono funzioni che agiscono su un singolo elemento del database;
- ***Query Functions***: sono funzioni che estraggono liste di elementi di un tipo;
- ***Utility Functions***: sono funzioni generiche che elaborano le liste.

Combinando insieme questi tre tipi di strumenti è stato possibile realizzare il tool di estrazione dei progetti di Statemate Magnum al fine di convertirli in formato XMI.

3 XML Metamodel Interchange (XMI)

3.1 XML ed XMI

L'XML (eXtensible Markup Language) è un linguaggio pensato per descrivere dati e modelli. A differenza dell'HTML non possiede tag propri: occorre definirli mediante l'uso di DTD (Data Type Definition). Un documento XML con il proprio DTD risulta essere autodescrittivo, non è necessario far riferimento a nessun altro elemento esterno. L'XML è stato creato per strutturare, memorizzare e scambiare informazioni: non ha caratteristiche operative, si limita esclusivamente a rappresentare l'informazione con specifiche regole sintattiche.

La specifica XMI (XML Metadata Interchange) è basata su W3C's Extensible Markup-Language (XML). Essa è composta principalmente da due componenti:

1. Regole di produzione di DTD XML per la creazione di Document Type Definitions per la codifica di modelli. Gli XMI DTDs sono utilizzati come specifiche sintattiche per la creazione di documenti XMI, e permettono la validazione XML di tali documenti.
2. Regole di produzione di documenti XML per la codifica di metadati in formato XML compatibile. Le regole di produzione possono essere usate per decodificare documenti XMI e ricostruire i metadati.

In sostanza l'XMI è un insieme di regole per la creazione di documenti XML che possano essere compresi da altri tools.

Esso pertanto consente di generare un DTD partendo da un metamodello, mediante l'applicazione di determinate regole di trasformazione.

Il metamodello deve essere descritto attraverso un framework di specifica per metadati chiamato MOF (Meta Object Facility), basato su un'architettura a quattro livelli standardizzata da OMG. Il metamodello UML è definito come uno di questi livelli, quindi è possibile utilizzare XMI per la produzione di un DTD per lo scambio di modelli UML.

3.2 Meta Object Facility (MOF)

MOF è la tecnologia adottata da OMG per la definizione e lo scambio di metadati. Nelle specifiche di MOF e XMI, un metadato è definito come un dato che descrive altri dati. Un modello è definito come una collezione di metadati correlati da regole che ne stabiliscono la struttura e la coerenza sintattica (sintassi astratta). Nella terminologia adottata da MOF, un metadato che descrive un altro metadato è chiamato meta-metadato. Un modello costituito da metadati è detto metamodello. MOF definisce una sintassi astratta comune per la definizione di metamodelli. Essa prende il nome di Modello MOF e si può considerare un modello per metamodelli, ossia un meta-metamodello. Inoltre MOF definisce un framework di meta-

modellazione basato su un'architettura a quattro livelli, schematizzata nella tabella 1. Descriviamo brevemente le caratteristiche e le responsabilità principali di ogni livello:

Tabella 1 - Architettura MOF a quattro livelli

Meta-livello	Termine MOF	Esempio
M3	meta-metamodello	*Modello MOF*
M2	meta-metadati	*Metamodello UML*
	metamodello	
M1	metadati	*Modello UML*
	modello	
M0	dati	*Sistema reale*
		database

***Meta-metamodello (M3)**:* costituisce l'infrastruttura per l'architettura di modellazione. La responsabilità principale di questo livello è definire un linguaggio per la specifica di metamodelli. Un meta-metamodello definisce un modello che si trova ad un livello di astrazione più alto rispetto ad un metamodello ed è generalmente più compatto dei metamodelli che descrive. Un meta-metamodello può definire molteplici metamodelli, inoltre possono esistere diversi meta-metamodelli associati ad un metamodello. Alcuni dei concetti tipici che si manipolano a questo livello sono quelli di MetaClasse, MetaAttributo e MetaOperazione.

***Metamodello (M2)**:* un metamodello è un'istanza di un meta-metamodello. Questo livello si occupa della definizione di un linguaggio per specificare modelli. Solitamente i metamodelli sono più complessi dei meta-metamodelli da cui sono descritti, specialmente quando definiscono una semantica dinamica. Alcuni concetti tipici di questo livello sono: Classe, Attributo, Operazione e Componente.

***Modello (M1)**:* un modello è un'istanza di un metamodello. La responsabilità principale di questo livello è definire un linguaggio che descrive un dominio di informazione. Un concetto tipico di questo livello è quello di una classe specifica, ad esempio la classe Point, dotata degli attributi x, y e delle operazioni setX e setY.

***Oggetti (M0)**:* sono le istanze dei modelli e servono a descrivere uno specifico dominio di informazione. Il concetto tipico di questo livello è quello di oggetto, inteso come istanza delle classi definite nel livello precedente.

Il *Modello MOF* è definito utilizzando se stesso come linguaggio di modellazione, poiché è posto al livello più alto. In altri termini, il Modello MOF è metamodello di se stesso.

MOF definisce tre concetti di base per la modellazione di metadati: Classe, Associazione e Package. Il loro significato è analogo a quello che ritroviamo nel linguaggio UML, con alcune semplificazioni:

Una *Classe* può comprendere *Attributi* e *Operazioni*, su cui è possibile definire vincoli di unicità e cardinalità. Inoltre, le Classi supportano l'ereditarietà multipla.

Le *Associazioni* sono sempre binarie e consentono di specificare l'ordinamento, il tipo di aggregazione, la cardinalità.

Un *Package* è una collezione di Classi e Associazioni. Inoltre, un Package può essere composto da altri Package.

Altri concetti chiave definiti da MOF sono quelli di DataType e Constraint. I primi vengono utilizzati per definire il tipo di un attributo, i secondi esprimono vincoli semantici che determinano la correttezza di un metamodello.

3.3 XMI e MOF

Poiché XMI viene proposto come un formato di scambio per metadati e MOF è la tecnologia adottata da OMG per rappresentare metadati, è naturale che l'architettura di riferimento su cui si basa XMI sia proprio MOF. Infatti, XMI rappresenta la tecnologia in grado di fornire un supporto per la *serializzazione di metamodelli* definiti mediante MOF, ossia un modo per scambiare metadati attraverso il semplice trasferimento di stringhe ASCII anziché mediante la definizione di interfacce di comunicazione basate, ad esempio, su CORBA.

XMI è costituito da un coppia di mappature parallele: una tra metamodelli MOF e DTD, l'altra tra metadati MOF e documenti XML. La prima mappatura è definita da un'insieme di regole che stabiliscono come ottenere in maniera automatica il DTD per un metamodello, partendo dalla sua rappresentazione MOF. Tali regole garantiscono che il DTD rispecchi il metamodello MOF di partenza, evitando qualsiasi tipo di errore.

Un DTD definisce la grammatica a cui un documento XML deve aderire per essere sintatticamente corretto. Mediante un DTD non è però possibile esprimere tutti i vincoli necessari a stabilire se un documento XML è semanticamente corretto. Di conseguenza, la verifica della correttezza semantica è compito di chi importa il documento, solitamente un tool specifico che deve conoscere il metamodello di riferimento.

3.4 Struttura di un DTD e di un documento XMI

La specifica di XMI definisce una collezione di regole che consentono di ottenere un DTD XML partendo da un metamodello MOF.

Nei paragrafi che seguono verranno descritte alcune delle regole di XMI e le caratteristiche comuni di ogni DTD ottenuto applicando quelle regole.

3.4.1 Rappresentazione di metaclassi

Una metaclasse di un metamodello viene rappresentata in XMI mediante un elemento avente lo stesso nome della metaclasse. Questo elemento è dotato di un serie di attributi che rispecchiano il nome e l'ordine di quelli della metaclasse che rappresenta. Un'associazione tra due metaclassi è rappresentata nel modo seguente: ciascuna delle due metaclassi ha un elemento figlio il cui nome equivale al ruolo dell'altra metaclasse e la cui molteplicità è quella definita dall'associazione.

3.4.2 Dichiarazioni iniziali

Tutti i DTD XMI condividono le seguenti dichiarazioni iniziali:

la versione di XML utilizzata e altre istruzioni XML valide, ad esempio:

```
<? XML version= “1.0” ?>
```

la dichiarazione del DTD di XMI, ad esempio:

```
<! DOCTYPE XMI SYSTEM “http://www.xmi.org/xmi.dtd” >
```

Ed altre dichiarazioni obbligatorie, descritte nel paragrafo successivo.

3.4.3 Dichiarazioni obbligatorie

Si trovano all'inizio di ogni DTD XMI e contengono informazioni riguardanti i metadati che si vogliono trasferire, come, ad esempio, il metamodello associato ai metadati, la data di creazione dei metadati, il nome del tool utilizzato per generare i metadati. Queste dichiarazioni sono contenute all'interno di un header indicato dall'elemento XMI.header e sono definite da elementi contraddistinti dal prefisso XMI, al fine di evitare conflitti con i nomi degli elementi che fanno parte del metamodello.

3.4.4 Attributi necessari

Sono gli attributi indispensabili in un DTD XMI. Si dividono in due categorie: quella per identificare gli elementi e quella per farvi riferimento:

***Attributi di identificazione**:*

Forniscono un identificatore univoco per un determinato elemento XMI. Ogni elemento che corrisponde ad una metaclasse di un metamodello deve avere almeno un attributo di identificazione.

Questi attributi sono tre (xmi.id, xmi.label, xmi.uuid) e vengono definiti nel modo seguente:

```
<!ENTITY % XMI.element.att
               'xmi.id ID #IMPLIED
                xmi.label CDATA #IMPLIED
                xmi.uuid CDATA #IMPLIED' >
```

`xmi.id`: ha lo scopo di associare un identificatore univoco ad un elemento. Viene utilizzato dagli attributi per riferimenti xmi.idref e href per fare riferimento ad uno specifico elemento.

`xmi.label`: è una stringa che può assumere qualsiasi valore, solitamente un identificatore scelto dall'utente.

`xmi.uuid`: viene utilizzato per assegnare ad un elemento un valore globalmente unico. UUID è l'acronimo di “Universally Unique IDentifier'' e rappresenta uno standard. È un identificatore a chiavi di 128 bit della forma “namespace:uuid'', ad esempio: DCE:2fac1234-31f8-11b4-a222-08002b34c003. Solitamente viene utilizzato nell'attributo href come XLink semplice.

Attributi per riferimenti:

Sono utilizzati all'interno di un elemento per fare riferimento ad un altro, utilizzando gli attributi di identificazione. Quindi un elemento può rappresentare un XLink semplice oppure può contenere un riferimento ad un altro elemento nello stesso documento. Questi attributi sono definiti nel modo seguente:

```
<!ENTITY % XMI.link.att
               'xml:link CDATA #IMPLIED
                href CDATA #IMPLIED
                xlink:inline (true | false) #IMPLIED
                xlink:actuate (show | user) #IMPLIED
                xlink:content-role CDATA #IMPLIED
                xlink:title CDATA #IMPLIED
                xlink:show (embed | replace | new) #IMPLIED
                xlink:behavior CDATA #IMPLIED
                xmi.idref IDREF #IMPLIED
                xmi.uuidref CDATA #IMPLIED' >
```

I primi otto attributi consentono di ottenere un XLink semplice. In particolare, occorre specificare in xmi:link il valore simple e in href il valore di un identificatore.

L'attributo xmi.idref consente di fare riferimento ad un elemento all'interno dello stesso documento. Quest'ultimo deve essere dotato dell'attributo xmi.id.

L'attributo xmi.uuidref consente di fare riferimento ad un altro elemento, anche appartenente ad un documento differente, utilizzando un UUID.

3.4.5 Elementi comuni ad ogni DTD

Ogni DTD XMI può includere la dichiarazione degli elementi seguenti:

- XMI;
- XMI.header;
- XMI.content;
- XMI.extensions;
- XMI.extension;
- XMI.documentation;
- XMI.owner;
- XMI.contact;
- XMI.longDescription;
- XMI.shortDescription;
- XMI.exporter;
- XMI.exporterVersion;
- XMI.exporterID;
- XMI.notice;
- XMI.model;
- XMI.metamodel;
- XMI.metametamodel;
- XMI.difference;
- XMI.delete;
- XMI.add;
- XMI.replace;
- XMI.reference;
- XMI.field;
- XMI.struct;
- XMI.seqItem;
- XMI.sequence;
- XMI.arrayLen;
- XMI.array;

- XMI.enum;
- XMI.discrim;
- XMI.union;
- XMI.any .

Si noti che questi elementi sono tutti preceduti dal prefisso XMI. Di seguito viene riportata una breve descrizione degli elementi più importanti.

XMI

È l'elemento radice di ogni documento XMI. La sua dichiarazione è la seguente:

```
<!ELEMENT XMI (XMI.header,
        XMI.content?,
        XMI.difference*,
        XMI.extensions*) >
<!ATTLIST XMI
        xmi.version CDATA #FIXED "1.1"
        timestamp CDATA #IMPLIED
        verified (true | false) #IMPLIED >
```

L'attributo xmi.version assume un valore costante e indica la versione della specifica XMI utilizzata.

L'attributo timestamp indica la data e l'ora in cui i metadati sono stati salvati. L'attributo verified indica se i metadati sono stati verificati: se viene impostato a true significa che il documento è semanticamente valido.

XMI.header

Contiene altri elementi che contengono informazioni sul modello, il metamodello e il metametamodello. La dichiarazione è la seguente:

```
<!ELEMENT XMI.header (XMI.documentation?,
        XMI.model*,
        XMI.metamodel*,
        XMI.metametamodel*) >
```

XMI.content

Contiene la definizione dei metadati, che possono rappresentare sia un modello che un metamodello. È dichiarato come segue:

```
<!ELEMENT XMI.content ANY >
```

XMI.documentation

Contiene informazioni riguardanti i metadati come l'autore, l'applicazione utilizzata per esportare i metadati con la relativa versione, note sul copyright ecc. La dichiarazione è la seguente:

```
<!ELEMENT XMI.documentation (#PCDATA |
        XMI.owner | XMI.contact |
        XMI.longDescription |
        XMI.shortDescription | XMI.exporter |
        XMI.exporterVersion | XMI.notice)* >
<!ELEMENT XMI.owner ANY >
<!ELEMENT XMI.contact ANY >
<!ELEMENT XMI.longDescription ANY >
<!ELEMENT XMI.shortDescription ANY >
<!ELEMENT XMI.exporter ANY >
<!ELEMENT XMI.exporterVersion ANY >
<!ELEMENT XMI.exporterID ANY >
<!ELEMENT XMI.notice ANY >
```

XMI.model

Identifica il modello di cui i dati trasferiti rappresentano una istanza. La dichiarazione è:

```
<!ELEMENT XMI.model ANY>
<!ATTLIST XMI.model
        xmi.name CDATA #REQUIRED
        xmi.version CDATA #REQUIRED >
```

Gli attributi xmi.name e xmi.version rappresentano rispettivamente il nome e la versione del modello.

XMI.metamodel

Identifica il metamodello relativo ai metadati trasmessi Conoscendo il metamodello, l'applicazione che importa i metadati può eseguire una verifica semantica su di essi. La dichiarazione è la seguente:

```
<!ELEMENT XMI.metamodel ANY>
<!ATTLIST XMI.metamodel
        %XMI.link.att;
        xmi.name CDATA #REQUIRED
        xmi.version CDATA #REQUIRED >
```

Come in precedenza, gli attributi xmi.name e xmi.version rappresentano il nome e la versione del metamodello.

XMI.metametamodel

Rappresenta il meta-metamodello a cui si conformano i metadati trasmessi. La dichiarazione è la seguente:

```
<!ELEMENT XMI.metametamodel ANY>
<!ATTLIST XMI.metametamodel
        %XMI.link.att;
```

```
        xmi.name CDATA #REQUIRED
        xmi.version CDATA #REQUIRED >
```

3.5 Specifica di un metamodello

Verrà descritto come vengono rappresentate le informazioni relative ad un metamodello in un DTD XMI. Un metamodello consiste essenzialmente in un diagramma delle classi, quindi occorre essere in grado di rappresentare le metaclassi e le relazioni tra esse.

3.5.1 Nomi qualificati

Un generico elemento di un metamodello deve avere un nome qualificato, che assume la forma seguente:

nome*1*.nome*2*. ... .nome*n* dove nome*1* è riferito al package più esterno, nome*n* a quello più interno.

3.5.2 Molteplicità

Le diverse molteplicità degli attributi appartenenti alle metaclassi definite nel metamodello, vengono rappresentate in un documento XML mediante simboli differenti. La Tabella 2 riassume i simboli utilizzati per ogni molteplicità.

Tabella 2 - Rappresentazione delle molteplicità definite in un metamodello

Molteplicità del metamodello	Molteplicità di XML
0 o 1	?
esattamente 1	' '
0 o più	*
1 o più	+
altro	*

3.5.3 Metaclassi

Ogni metaclasse di un metamodello è composta da tre parti, ognuna delle quali viene rappresentata mediante un'entità diversa. Indichiamo di seguito il nome di ogni parte e le caratteristiche dell'entità che la rappresenta:

Proprietà: l'entità corrispondente deve contenere una lista di elementi corrispondenti agli attributi del metamodello.

Associazioni: la relativa entità contiene gli elementi che rappresentano il ruolo della classe nelle associazioni diverse dalle aggregazioni.

Composizioni: la relativa entità contiene una lista di elementi che rappresentano il ruolo della classe nelle aggregazioni.

Il nome delle tre entità inizia con quello classe e termina con uno dei seguenti suffissi: *Properties Associations Compositions.*

Ad esempio, consideriamo una classe chiamata *Point*, che contiene attributi, associazioni ed aggregazioni. La sua dichiarazione è la seguente:

```
<!ENTITY % PointProperties 'propertiesForPoint'>
<!ENTITY % PointAssociations 'associationsForPoint'>
<!ENTITY % PointCompositions 'XMI.extension*,
          compositionsForPoint'>
<!ELEMENT Point (%PointProperties, %PointAssociations,
            %PointCompositions)? >
<!ATTLIST Point
        %XMI.element.att;
        %XMI.link.att; >
```

3.5.4 Ereditarietà

Poiché XML non definisce un meccanismo per rappresentare l'ereditarietà, in XMI viene definita copiando tutti gli attributi, le associazioni e le aggregazioni da una superclasse alle sue sottoclassi.

Ad esempio, consideriamo la classe Point, che specializza la classe Shape. Le entità di Point vengono definite come segue:

```
<!ENTITY % PointProperties '%ShapeProperties ;
                        properties for Point' >
<!ENTITY % PointAssociations '%ShapeAssociations;
                          associations for Point' >
<!ENTITY % PointCompositions '%ShapeCompositions;
                          compositions for Point' >
```

3.5.5 Attributi

Gli attributi vengono rappresentati mediante elementi XML. Ogni dichiarazione ha la forma seguente:

```
<!ELEMENT a (type specification) >
```

Ad esempio, gli attributi x e y (di tipo intero) della classe Point vengono dichiarati come segue:

```
<!ELEMENT x (#PCDATA | XMI.reference) *>
<!ELEMENT y (#PCDATA | XMI.reference) * >
<!ENTITY % PointProperties 'x, y' >
```

dove XMI.reference rappresenta un riferimento al tipo di dato intero. La molteplicità degli attributi viene espressa secondo le regole della Tabella 2. Ad esempio, se le molteplicità degli attributi x e y fossero rispettivamente di uno o più e di zero o più, allora la dichiarazione sarebbe del tipo:

```
<!ENTITY % PointProperties 'x+, y*' >
```

3.5.6 Associazioni

Ogni associazione è rappresentata da un'entità e da un elemento XML. La molteplicità del ruolo svolto dalla classe che partecipa all'associazione, deve rispettare le regole definite nella Tabella 2.

Ad esempio, un'associazione di ruolo r con molteplicità m per la metaclasse Point è dichiarata nel modo seguente:

```
<!ENTITY % PointAssociations 'rm' >
<!ELEMENT r (content)? >
```

dove content rappresenta l'insieme degli elementi relativi alle classi con cui Point stabilisce un'associazione.

3.5.7 Composizioni

Una composizione è un particolare tipo di associazione. Se la classe Point partecipa ad una relazione di composizione con la classe Shape di ruolo r e molteplicità m, la dichiarazione è la seguente:

```
<!ELEMENT r (Shape)? >
<!ENTITY % PointCompositions 'XMI.extension*', 'rm' >
```

3.5.8 Meccanismi di linking

XMI mette a disposizione diversi meccanismi per specificare riferimenti all'interno dello stesso documento e tra documenti diversi. Le caratteristiche e i principi su cui si basano questi meccanismi sono i seguenti:

- i link si basano su XLink per navigare tra documenti diversi e all'interno dello stesso documento;
- i link hanno la stessa forma sia che si referenzi un elemento nello stesso documento sia che si punti ad un documento esterno;
- la definizione di un link è incapsulata nell'entità XMI.link.att.
- l'entità XMI.link.att supporta link esterni attraverso gli attributi di XLink e link interni attraverso gli attributi xmi.id e xmi.uuid.

3.6 XMI e UML

Il metamodello UML è definito nella specifica del linguaggio come un metamodello MOF. Quindi è possibile applicare le regole di generazione di un DTD XMI alla specifica del linguaggio UML, al fine di ottenere il DTD relativo al metamodello UML.

3.6.1 Struttura del DTD di UML

La prima parte del DTD di UML dichiara gli elementi comuni a tutti i DTD. Nella seconda parte, molto più ampia, sono definiti gli elementi che costituiscono il metamodello UML.

Gli elementi del metamodello utilizzano nomi qualificati che si rifanno alla divisione in package del metamodello UML. Ad esempio, l'elemento che definisce una classe è il seguente:

```
<!ELEMENT Foundation.Core.Class (...)?>
```

Infatti, nel metamodello UML l'elemento Class è definito nel package Core, subpackage del package Foundation.

Gli attributi di una classe del metamodello UML sono definiti come elementi appartenenti al content model dell'elemento relativo alla classe. L'esempio che segue mostra la dichiarazione della classe ModelElement, dotata dell'attributo name:

```
<!ELEMENT Foundation.Core.ModelElement.name
          (#PCDATA | XMI.reference)* >
          ...
<!ELEMENT Foundation.Core.ModelElement
          (Foundation.Core.ModelElement.name,
          ...)?>
```

Poiché in XML non è definito un meccanismo di ereditarietà tra elementi, la dichiarazione di una classe che eredita determinate caratteristiche da un'altra classe deve ripetere l'inclusione delle caratteristiche in questione.

```
<!ELEMENT Foundation.Core.Class
          (Foundation.Core.ModelElement.name,
          ...)?>
```

Nell'esempio precedente si dichiara l'elemento Foundation.Core.Class. Nel metamodello UML l'elemento Class eredita da ModelElement l'attributo name. Per questo motivo nel content model dell'elemento dichiarato ritroviamo l'elemento Foundation.Core.ModelElement.name. In questo caso il nome qualificato ci aiuta a risalire al fatto che, nel metamodello, l'attributo name è dichiarato in ModelElement.

3.7 Conclusioni su XMI

XMI è una piattaforma comune per lo scambio di metadati basato su XML. Il suo punto di forza consiste nell'essere un formato aperto, costruito su una serie di standard consolidati quali XML e MOF e dal suo essere indipendente dalla piattaforma.

4 Analisi ed implementazione del software

In un primo momento si è pensato di utilizzare il metamodello XMI per l'UML proposto dall'OMG per l'esportazione dei diagrammi di Harel da StateMate, successivamente ci si è resi conto che le differenze tra State-Charts UML e State-Charts di Harel in alcuni casi sono sostanziali. Ad esempio in UML non esistono i connectors, ed anche se gli pseudo-stati permettono la creazione di modelli equivalenti a quelli di Harel, i singoli elementi non hanno sempre la stessa semantica. Altre differenze si riscontrano nella gestione degli eventi che in UML sono associati ad un modello, in StateMate sono associati alle Statechart a loro volta contenute in un progetto (modello). In sostanza non è possibile esportare un modello di StateMate in UML mantenendone l'esatta struttura.

Si è pertanto creato un modello XML, rispettando le specifiche XMI, che descrivesse le Statechart di StateMate.

L'esportazione in XMI dei diagrammi di Statemate comporta pertanto la generazione di un DTD che rispetti le regole di Harel.

4.1 Harel DTD

Per creare il DTD abbiamo seguito la struttura di quello proposto dall'OMG per le macchine a stati (XMI 1.2), riportando solo gli elementi necessari al modello in esame. Abbiamo rinominato alcuni elementi (per facilitare la lettura), aggiunto quelli mancanti e alterato la struttura laddove non rispecchiava quella di Statemate. Il DTD proposto dall'OMG risulta essere ridondante, nel senso che le informazioni possono essere salvate in più parti, e in più modi, noi abbiamo limitato tale ripetitività. Per chiarire questo aspetto di seguito l'elemento ***state*** è rappresentato secondo OMG e secondo la struttura priva di ridondanza.

***State* secondo OMG**

```
<!ELEMENT UML:State (UML:ModelElement.name |
UML:ModelElement.visibility | XMI.extension |

...

* >

<!ATTLIST UML:State
name CDATA #IMPLIED
visibility (public | protected | private) #IMPLIED

...

%XMI.element.att;
%XMI.link.att;
>
```

***State* definito in Harel.dtd**

```
<!ELEMENT HAREL:State (XMI.extension |      HAREL:StateVertex.outgoing |
 HAREL:StateVertex.incoming | HAREL:StateVertex.parent |
 HAREL:State.stateChart | HAREL:State.Reaction)*
>

<!ATTLIST HAREL:State
 name CDATA #IMPLIED
 %XMI.element.att;
 %XMI.link.att;
>
```

Come si può vedere, il nome di un elemento in Harel.dtd è nella sua lista degli attributi, per OMG può anche essere nella lista dei parametri.

Per quanto riguarda il DTD, in appendice A è riportata la definizione del DTD (solo per la parte relativa agli StateCharts) per i modelli di Harel, mentre qui di seguito sono riportate le definizioni degli elementi necessari al DTD per l'esportazione delle activity-chart (analoghe a quelle per gli statechart come in appendice A). Sono state quindi definite le activity-chart e i relativi sotto-elementi, quali le flow-lines e i data-store.

```
<!--___________ HAREL:StateMachines.ActivityChart_________________ -->
<!ELEMENT HAREL:ActivityChart.context (HAREL:Link)*>
<!ELEMENT HAREL:ActivityChart.Events (HAREL:Event)*>
<!ELEMENT   HAREL:ActivityChart   (XMI.extension   |   HAREL:ActivityChart.context   |
HAREL:ActivityChart.Activities        |        HAREL:ActivityChart.Flowlines        |
HAREL:ActivityChart.Events | HAREL:Note)*>
<!ATTLIST HAREL:ActivityChart
 name CDATA #IMPLIED
 %XMI.element.att;
 %XMI.link.att; >
```

```
<!--____ HAREL:StateMachines.ActivityChart.Activities  ___________ -->
<!ELEMENT HAREL:ActivityChart.Activities.SubChart (HAREL:Link)*>
<!ELEMENT   HAREL:ActivityChart.Activities   (HAREL:Activity   |   HAREL:Datastore   |
HAREL:Connector | HAREL:Instance | HAREL:ActivityChart.Activities.SubChart)*>
```

```
<!-- _________ HAREL:StateMachines.ActivityChart.Flowlines _________ -->
<!ELEMENT HAREL:ActivityChart.Flowlines (HAREL:Flowline)*>
```

```
<!-- _____________ HAREL:StateMachines.Flowline _____________ -->
<!ELEMENT HAREL:Flowline.source (HAREL:ActivityVertex)*>
<!ELEMENT HAREL:Flowline.target (HAREL:ActivityVertex)*>
<!ELEMENT HAREL:Flowline.activitychart (HAREL:ActivityChart)*>
<!ELEMENT HAREL:Flowline (XMI.extension | HAREL:Expression | HAREL:Flowline.source
| HAREL:Flowline.target | HAREL:Flowline.activitychart)*>
<!ATTLIST HAREL:Flowline
 type CDATA #IMPLIED
 %XMI.element.att;
 %XMI.link.att; >
```

```
<!-- _____________ HAREL:StateMachines.ActivityVertex _____________ -->
<!ELEMENT HAREL:ActivityVertex.outgoing (HAREL:Flowline)*>
<!ELEMENT HAREL:ActivityVertex.incoming (HAREL:Flowline)*>
<!ELEMENT HAREL:ActivityVertex.parent (HAREL:NonBasicActivity)*>
<!ELEMENT  HAREL:ActivityVertex  (XMI.extension  |  HAREL:ActivityVertex.outgoing  |
HAREL:ActivityVertex.incoming | HAREL:ActivityVertex.parent)*>
<!ATTLIST HAREL:ActivityVertex
 name CDATA #IMPLIED
 %XMI.element.att;
 %XMI.link.att;>
```

```
<!-- _____________ HAREL:StateMachines.Activity_____________ -->
<!ELEMENT HAREL:Activity.Reaction (HAREL:Expression)*>
<!ELEMENT HAREL:Activity.activityChart (HAREL:ActivityChart)*>
<!ELEMENT   HAREL:Activity   (XMI.extension   |   HAREL:ActivityVertex.outgoing   |
HAREL:ActivityVertex.incoming        |        HAREL:ActivityVertex.parent        |
HAREL:Activity.activityChart         |          HAREL:Activity.Reaction          |
HAREL:NonBasicActivity.SubVertex)*>
<!ATTLIST HAREL:Activity
 name CDATA #IMPLIED
 type CDATA #IMPLIED
 %XMI.element.att;
 %XMI.link.att;>
```

```
<!-- _____________ HAREL:StateMachines.Datastore_____________ -->
<!ELEMENT   HAREL:Datastore   (XMI.extension   |   HAREL:ActivityVertex.outgoing   |
HAREL:ActivityVertex.incoming        |        HAREL:ActivityVertex.parent        |
HAREL:Activity.activityChart)*>
<!ATTLIST HAREL:Datastore
 name CDATA #IMPLIED
 %XMI.element.att;
 %XMI.link.att;>
```

4.2 Struttura del programma

Il funzionamento del programma è suddiviso in due fasi distinte e consecutive: la raccolta dati e la creazione del codice XMI. Entrambe queste fasi sono indipendenti l'una dall'altra anche per quanto riguarda le funzioni utilizzate; l'unico punto di contatto è rappresentato dalla struttura dati salvata in memoria.

Nella prima fase tale struttura verrà popolata con i dati provenienti dal database, nella seconda essa rappresenterà l'unica fonte di informazione utile per la creazione dell'output XMI. Le due parti in cui il programma si divide sono quindi:

1. la raccolta dati dal database di Statemate tramite Dataport e l'inserimento di tali dati in apposite strutture C;
2. la lettura dei dati da tali strutture C e la creazione dell'output in formato XMI.

Inizialmente il programma accede al database di Statemate attraverso query ad hoc che consentono di memorizzare le informazioni necessarie nelle strutture descritte precedentemente.

4.3 Strutture di memorizzazione implementate

Nella stesura del codice XMI è preferibile avere a disposizione dei dati strutturati in modo da facilitare il recupero dei dati di interesse. Nel file xmi_exporter.h è possibile trovare la definizione delle strutture relative alla memorizzazione dei dati delle activity-chart e degli statechart.

Le principali strutture usate sono:

- PROJECT;
- STATECHART;
- STATE;
- TRANSITION;
- EVENT;
- ACTIVITYCHART;
- ACTIVITY;
- FLOWLINE.

Riportiamo di seguito la loro definizione:

```
Typedef structs_project {
char*name;
XMID xmi_id;
PTR_LIST*statechart_list;
} PROJECT;
```

name: nome del progetto

xmi_id: XMI id del progetto

statechart_list: lista delle statechart presenti nel progetto

---

```
typedefstructs_chart{
char*name;
stm_idchart_id;
XMIDxmi_id;
XMIDxmi_top_id;
PTR_LIST*top_state_list;
PTR_LIST*transition_list;
PTR_LIST*event_list;
} STATECHART;
```

name: nome della statechart

chart_id: id associato da Statemate alla statechart

xmi_id: XMI id della statechart

xmi_top_id: XMI id del parent topstate

list: lista degli stati direttamente contenuti nella statechart

transition_list: lista delle transizioni contenute dalla statechart

event_list: lista degli eventi definiti nella statechart

```
typedef structs_state{
char*name;
stm_idstate_id;
XMIDxmi_id;
XMIDparent_id;
PARENT_TYPEparent_type;

STATE_TYPEtype;
boolean pseudo;
STATECHART*instance;

char*reactions;

PTR_LIST*incoming_transition_list;
PTR_LIST*outgoing_transition_list;

PTR_LIST*child_list;
}STATE;
```

name: nome dello stato

state_id: id associato da Statemate allo stato

xmi_id: XMI id dello stato

parent_id: XMI id del parent

parent_type: type del parent (StateChart o CompositeState)

type: type dello stato (AND, OR, BASIC, JUNCTION....)

pseudo: true se è un connector (pseudo stato)

instance: Se lo stato è un instance, tale variabile contiene il riferimento alla StateChart di cui è instance

reactions: espressione contenente trigger, guard-condition e action per la gestione della lista delle azioni da compiere all'ingresso nello stato

incoming_transition_list: lista delle transizioni (frecce) che arrivano a questo a stato

outgoing_transition_list: lista delle transizioni (frecce) che escono da questo stato

child_list: contiene la lista degli stati in esso contenuti se lo stato è un CompositState

---

```
typedefstructs_transition{
stm_idtransition_id;
XMIDxmi_id;
XMIDchart_id;
char*expression;
structs_state*source;
structs_state*target;

} TRANSITION;
```

transition_id: id associato da Statemate alla transition

xmi_id: XMI id della transition

chart_id: XMI id della chart che la contiene

expression: espressione contenente trigger, guard-condition e action

source: stato da cui parte la transition

target: stato a cui arriva la transition

```
typedefstructs_event{
stm_idevent_id;
char*name;
XMIDchart_id;
XMIDxmi_id;
char*expression;

}EVENT;
```

event_id: id associato da Statemate all'evento

name: nome dell'evento

chartid: XMI id della chart in cui è contenuto l'evento

xmi_id: XMI id dell'evento

expression: espressione dell'evento

```
typedef struct a_chart{
 char* name;
 stm_id chart_id;
 XMID xmi_id;
 XMID xmi_top_id;
 PTR_LIST* top_activity_list;
 PTR_LIST* flowline_list;
 PTR_LIST* connector_list;
 PTR_LIST* event_list;
 PTR_LIST* note_list;
} ACTIVITYCHART;
```

name: nome dell'activitychart;

chart_id: id associato da Statemate all'activitychart;

xmi_id: XMI id dell'activitychart;

xmi_top_id: XMI id del *parent;*

top_activity_list: lista delle activitychart contenute all'interno;

flowline_list: lista delle flowline contenute;

connector_list: lista dei connettori contenuti;

event_list: lista degli eventi contenuti;

note_list: lista delle note contenute.

```
typedef struct a_activity{
 stm_id activity_id;
 XMID xmi_id;
 XMID parent_id;
 ACT_PARENT_TYPE parent_type;
 char* name;
 ACTIVITY_TYPE type;
 boolean datastore;
 boolean connector;
 ACTIVITYCHART* instance;

 char* reactions;

 PTR_LIST* incoming_flowline_list;
 PTR_LIST* outgoing_flowline_list;

 PTR_LIST* child_list;
} ACTIVITY;
```

name: nome della activity;

activity_id: id associato da Statemate alla activity;

xmi_id: XMI id della activity;

parent_id: XMI id del parent;

parent_type: tipo del parent;

type: tipo della activity;

datastore: indica se l'activity in questione è un datastore (pseudo-activity);

connector: indica se l'activity in questione è in realtà un connettore (pseudo-activity);

instance: se l'activity è di tipo Instance, rappresenta il rif. All'ActivityChart padre;

reactions: espressione contenente trigger, guard-condition e action;

incoming_flowline_list: lista delle flowline entranti;

outgoing_flowline_list: lista delle flowline uscenti;

child_list: contiene la lista delle eventuali activity contenute.

```
typedef struct a_flowline{
 stm_id flowline_id;
 XMID xmi_id;
 XMID chart_id;
 FLOWLINE_TYPE type;
 char* expression;
 struct a_activity* source;
 struct a_activity* target;
} FLOWLINE;
```

flowline_id: id associato da Statemate alla flowline;

xmi_id: XMI id della flowline;

chart_id: XMI id della chart che contiene la flowline;

type: indica la tipologia della flowline (dati o controlli);

expression: espressione contenente trigger, guard-condition e action;

source: activity da cui parte la flowline;

target: activity a cui arriva la flowline.

4.3.1 Accesso al database di Statemate e raccolta dati

Prima di accedere al database di Statemate è necessario inizializzarlo tramite la funzione *stm_init_uadche*, la quale ha tra i parametri in ingresso la workarea ed il nome del progetto.

Una volta inizializzato il database, si possono usare le query functions per ottenere i dati voluti. Le funzioni principali da noi implementate per prendere i dati dal database e metterli nelle apposite strutture sono:

VisitaStateCharts: trova tutte le statechart della work-area e le salva nella struttura PROJECT;

VisitaStates: trova tutti gli stati e i connectors, quindi li salva (nelle rispettive statechart) rispettandone l'annidamento;

VisitaTransitions: trova tutte le transition e le salva nelle rispettive state-chart;

VisitaEvents: trova tutti gli event e li salva nelle rispettive statechart.

Per quel che riguarda le activity-charts le principali funzioni implementate sono:

VisitaActivityCharts: scandisce le activity-chart dell'area di lavoro e le salva nella struttura project;

VisitaActivities: scandisce tutte le attività (quindi la top-activity, le external activities, i data-store e i connettori) e le memorizza rispettandone l'ordine di annidamento;

VisitaFlowlines: scandisce tutte le flow-line della activity-chart e dei sottoelementi e le memorizza;

VisitaEvents: visita gli eventi e li memorizza;

VisitaNotes: visita le note e le memorizza.

4.3.2 Creazione del file XMI

La funzione principale che crea l'XMI a partire dal PROJECT è CreaXMI. Essa legge i dati memorizzati in fase di acquisizione e utilizzando funzioni di supporto per la creazione dei vari elementi XMI, genera l'output richiesto. La struttura dell'output generato si basa su due DTD: il DTD standard dell'XMI 1.2 e quello esteso (Harel.dtd) per descrivere i diagrammi di Harel usati da Statemate.

5 Output XMI dell'applicazione xmi_exporter

Sono riportati di seguito tre modelli di sistema e il relativo output XMI prodotto dall'applicazione sviluppata.

5.1 Esempio 1

Vediamo ora un semplice modello di macchina a stati realizzato con StateMate (Figura 2 – Esempio di Statechart) e del relativo output generato dal tool (XMI Exporter):

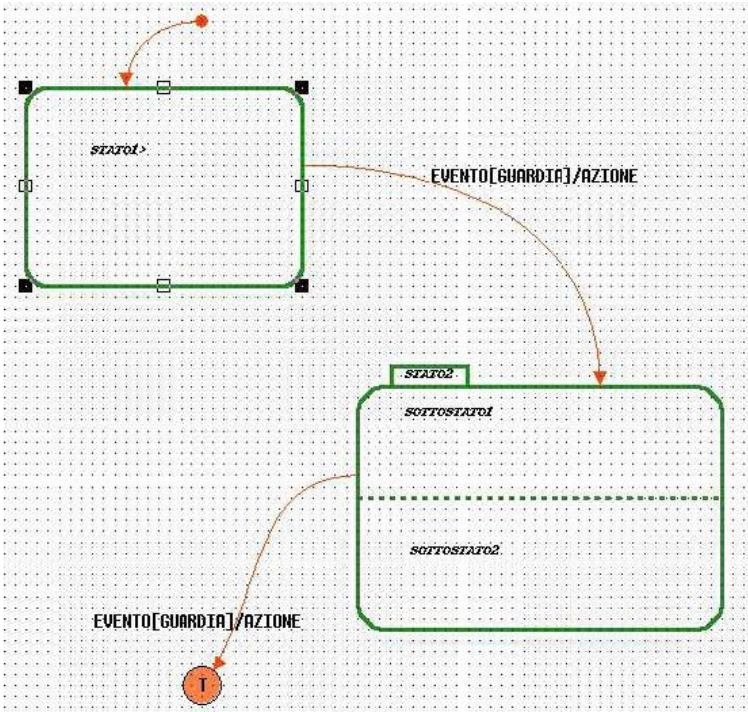

Figura 2 – Esempio di Statechart

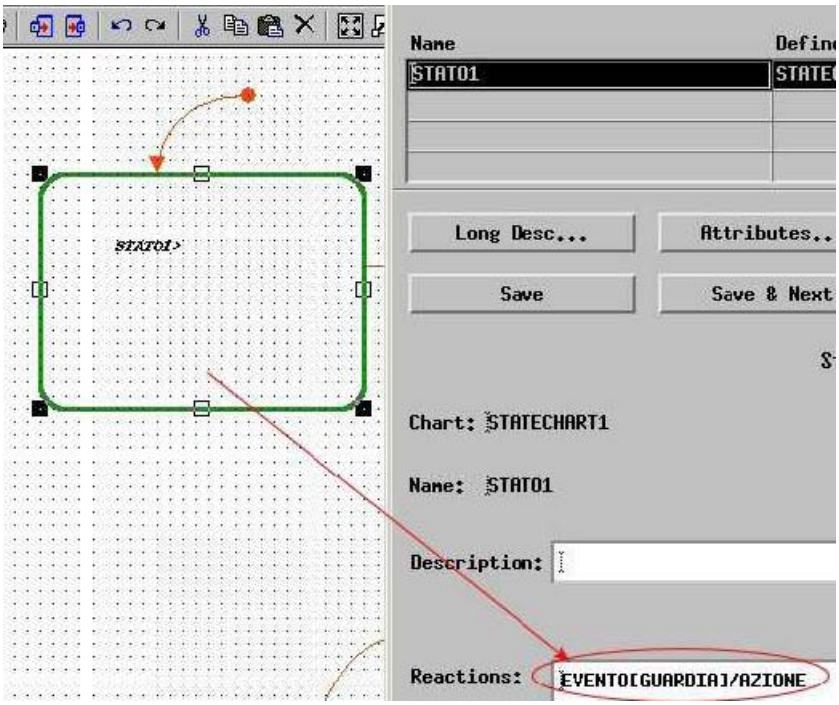

Figura 3 – Reactions dello STATO1

```
<?xml version="1.0" encoding="UTF-8"?> <!DOCTYPE XMI PUBLIC "Harel" "./harel.dtd">
<XMI xmi.version="1.2">
  <XMI.header>
    <XMI.documentation>
      <XMI.exporter>xmi_exporter</XMI.exporter>
      <XMI.exporterVersion>1.0</XMI.exporterVersion>
    </XMI.documentation>
    <XMI.metamodel xmi.name="Harel.dtd" xmi.version="1.3"/>
  </XMI.header>
  <XMI.content>
    <HAREL xmlns:HAREL='./Harel.dtd'>
      <HAREL:Project name="PROGETTO" xmi.id="xmi.2">
        <HAREL:ownedElement>
          <HAREL:StateChart name="STATECHART1" xmi.id="xmi.3">
            <HAREL:StateChart.context>
              <HAREL:Link xmi.idref="xmi.2"/>
            </HAREL:StateChart.context>
            <HAREL:StateChart.States>
              <HAREL:State name="STATO1" xmi.id="xmi.5">
                <HAREL:State.stateChart>
                  <HAREL:StateChart xmi.idref="xmi.3"/>
                </HAREL:State.stateChart>
                <HAREL:State.Reaction>
                  <HAREL:Expression>EVENTO[GUARDIA]/AZIONE</HAREL:Expression>
                </HAREL:State.Reaction>
                <HAREL:StateVertex.outgoing>
                  <HAREL:Transition xmi.idref="xmi.11"/>
                </HAREL:StateVertex.outgoing>
                <HAREL:StateVertex.incoming>
                  <HAREL:Transition xmi.idref="xmi.12"/>
                </HAREL:StateVertex.incoming>
```

```
              </HAREL:State>
              <HAREL:CompositeState name="STATO2" isConcurrent="true"
xmi.id="xmi.6">
                <HAREL:State.stateChart>
                  <HAREL:StateChart xmi.idref="xmi.3"/>
                </HAREL:State.stateChart>
                <HAREL:State.Reaction>
                  <HAREL:Expression></HAREL:Expression>
                </HAREL:State.Reaction>
                <HAREL:StateVertex.outgoing>
                  <HAREL:Transition xmi.idref="xmi.13"/>
                </HAREL:StateVertex.outgoing>
                <HAREL:StateVertex.incoming>
                  <HAREL:Transition xmi.idref="xmi.11"/>
                </HAREL:StateVertex.incoming>
                <HAREL:CompositeState.SubVertex>
                  <HAREL:State name="SOTTOSTATO1" xmi.id="xmi.7">
                    <HAREL:StateVertex.parent>
                      <HAREL:CompositeState xmi.idref="xmi.6"/>
                    </HAREL:StateVertex.parent>
                    <HAREL:State.Reaction>
                      <HAREL:Expression></HAREL:Expression>
                    </HAREL:State.Reaction>
                  </HAREL:State>
                  <HAREL:State name="SOTTOSTATO2" xmi.id="xmi.8">
                    <HAREL:StateVertex.parent>
                      <HAREL:CompositeState xmi.idref="xmi.6"/>
                    </HAREL:StateVertex.parent>
                    <HAREL:State.Reaction>
                      <HAREL:Expression></HAREL:Expression>
                    </HAREL:State.Reaction>
                  </HAREL:State>
                </HAREL:CompositeState.SubVertex>
              </HAREL:CompositeState>
              <HAREL:Connector value="" xmi.id="xmi.9">
                <HAREL:State.stateChart>
                  <HAREL:StateChart xmi.idref="xmi.3"/>
                </HAREL:State.stateChart>
                <HAREL:Connector.kind xmi.value="default" />
                  <HAREL:StateVertex.outgoing>
                    <HAREL:Transition xmi.idref="xmi.12"/>
                  </HAREL:StateVertex.outgoing>
                </HAREL:Connector>
                <HAREL:Connector value="" xmi.id="xmi.10">
                  <HAREL:State.stateChart>
                    <HAREL:StateChart xmi.idref="xmi.3"/>
                  </HAREL:State.stateChart>
                  <HAREL:Connector.kind xmi.value="termination" />
                    <HAREL:StateVertex.incoming>
                      <HAREL:Transition xmi.idref="xmi.13"/>
                    </HAREL:StateVertex.incoming>
                  </HAREL:Connector>
                </HAREL:StateChart.States>
                <HAREL:StateChart.Transitions>
                  <HAREL:Transition xmi.id="xmi.11">
                    <HAREL:Transition.statechart>
                      <HAREL:StateChart xmi.idref="xmi.3"/>
                    </HAREL:Transition.statechart>
                    <HAREL:Expression>EVENTO[GUARDIA]/AZIONE</HAREL:Expression>
                      <HAREL:Transition.source>
                        <HAREL:StateVertex xmi.idref="xmi.5"/>
                      </HAREL:Transition.source>
                      <HAREL:Transition.target>
```

```
                    <HAREL:StateVertex xmi.idref="xmi.6"/>
                  </HAREL:Transition.target>
                </HAREL:Transition>
                <HAREL:Transition xmi.id="xmi.12">
                  <HAREL:Transition.statechart>
                    <HAREL:StateChart xmi.idref="xmi.3"/>
                  </HAREL:Transition.statechart>
                  <HAREL:Expression></HAREL:Expression>
                  <HAREL:Transition.source>
                    <HAREL:StateVertex xmi.idref="xmi.9"/>
                  </HAREL:Transition.source>
                  <HAREL:Transition.target>
                    <HAREL:StateVertex xmi.idref="xmi.5"/>
                  </HAREL:Transition.target>
                </HAREL:Transition>
                <HAREL:Transition xmi.id="xmi.13">
                  <HAREL:Transition.statechart>
                    <HAREL:StateChart xmi.idref="xmi.3"/>
                  </HAREL:Transition.statechart>
                  <HAREL:Expression>EVENTO[GUARDIA]/AZIONE</HAREL:Expression>
                  <HAREL:Transition.source>
                    <HAREL:StateVertex xmi.idref="xmi.6"/>
                  </HAREL:Transition.source>
                  <HAREL:Transition.target>
                    <HAREL:StateVertex xmi.idref="xmi.10"/>
                  </HAREL:Transition.target>
                </HAREL:Transition>
              </HAREL:StateChart.Transitions>
              <HAREL:StateChart.Events>
                <HAREL:Event name="AZIONE" xmi.id ="xmi.14" />
                  <HAREL:Event name="EVENTO" xmi.id ="xmi.15" />
                </HAREL:StateChart.Events>
              </HAREL:StateChart>
            </HAREL:ownedElement>
          </HAREL:Project>
        </HAREL>
      </XMI.content>
    </XMI>
    <HAREL:StateChart.Events>
      <HAREL:Event name="AZIONE" xmi.id ="xmi.14" />
      <HAREL:Event name="EVENTO" xmi.id ="xmi.15" />
    </HAREL:StateChart.Events>
  </HAREL:StateChart>
</HAREL:ownedElement>
</HAREL:Project>
</HAREL>
</XMI.content>
</XMI>
```

5.1.1 Analisi dell'output

Come si può vedere la struttura dell'output rispecchia la struttura della StateChart ed è la seguente:

- Intestazioni xml varie.
- `<XMI.header>`: questo è l'header XMI dove ci sono varie informazioni, tra cui il metamodello utilizzato, il nome dell'exporter, etc...

- <XMI.content>: qui dentro vanno i vari elementi del modello.
- <HAREL:Projectname="PROGETTO"xmi.id="xmi.2">: include il tag <HAREL:ownedElement> che a sua volta contiene tutti gli elementi del progetto (ad esempio le StateChart).
- <HAREL:StateChartname="STATECHART1"xmi.id="xmi.3">: questo tag rappresenta una StateChart. Essa può avere al suo interno la lista degli stati, le transition, gli event.
- <HAREL:StateChart.States>: lista degli stati. A sua volta alcuni stati (CompositeState) possono contenere altri stati.
- <HAREL:StateChart.Transitions>: lista delle transition.
- <HAREL:StateChart.Events>: lista degli eventi.

Ogni elemento possiede un id univoco (xmi.id) che è essenziale per far riferimento ai singoli oggetti. Ad esempio nelle transition si fa riferimento allo stato source ed allo stato target. All'interno di <HAREL:StateChart.States> ci possono essere State, CompositeState e Connector. I CompositeState presentano un attributo "isConcurrent" il quale prende il valore di "true" se gli stati che contiene sono concorrenti. I Connector invece presentano al loro interno un tag <HAREL:Connector.kindxmi.value="tipodelconnector"/> che permette di specificare il suo tipo (ad esempio initial, termination,...). Nel codice possiamo notare delle informazioni ripetute, ad esempio negli stati vi è una lista di transition entranti ed una di transition uscenti; **nelle transition vi è il riferimento allo stato di partenze (source) e allo stato di arrivo (target).**

5.2 Esempio 2

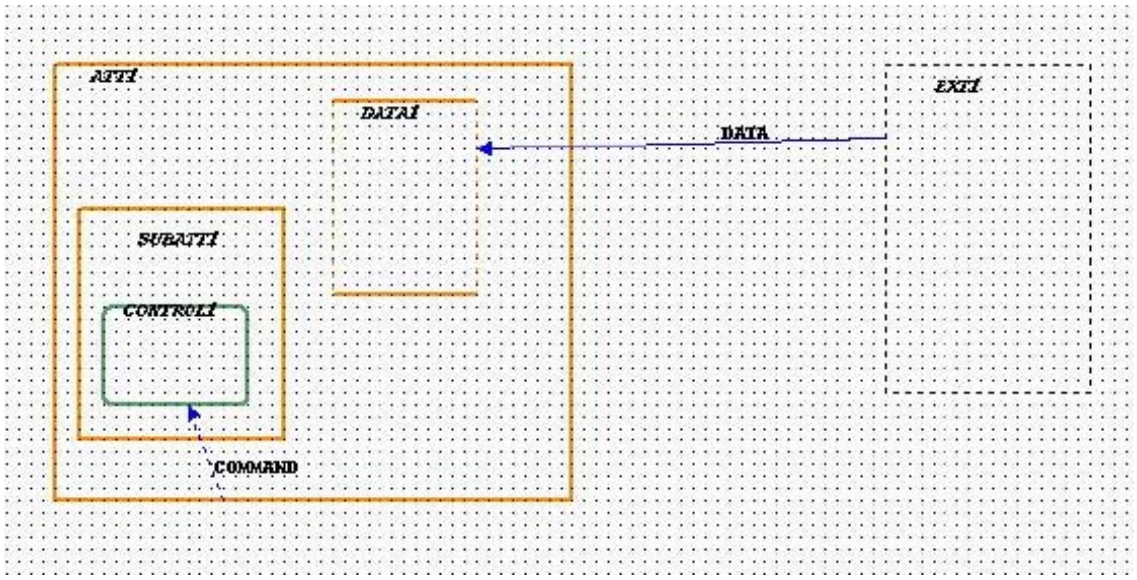

Figura 4 – Esempio 1: esportazione di Activity Charts e Statecharts

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE XMI PUBLIC "Harel" "./harel.dtd">

<XMI xmi.version="1.2">
  <XMI.header>
    <XMI.documentation>
      <XMI.exporter>xmi_exporter</XMI.exporter>
      <XMI.exporterVersion>2.0</XMI.exporterVersion>
    </XMI.documentation>
    <XMI.metamodel xmi.name="Harel.dtd" xmi.version="1.3"/>
  </XMI.header>
  <XMI.content>
    <HAREL xmlns:HAREL='./Harel.dtd'>
      <HAREL:Project name="" xmi.id="xmi.2">
        <HAREL:ownedElement>
          <HAREL:ActivityChart name="ACT_TEST1" xmi.id="xmi.3">
            <HAREL:ActivityChart.context>
              <HAREL:Link xmi.idref="xmi.2"/>
            </HAREL:ActivityChart.context>
            <HAREL:ActivityChart.Activities>
              <HAREL:Activity name="ATT1" type="INTERNAL" xmi.id="xmi.5">
                <HAREL:Activity.activityChart>
                  <HAREL:ActivityChart xmi.idref="xmi.3"/>
                </HAREL:Activity.activityChart>
                <HAREL:Activity.Reaction>
                  <HAREL:Expression></HAREL:Expression>
                </HAREL:Activity.Reaction>
                <HAREL:ActivityVertex.outgoing>
                  <HAREL:Flowline xmi.idref="xmi.11"/>
                </HAREL:ActivityVertex.outgoing>
                <HAREL:NonBasicActivity.SubVertex>
                  <HAREL:Datastore name="DATA1" xmi.id="xmi.6">
                    <HAREL:ActivityVertex.parent>
                      <HAREL:NonBasicActivity xmi.idref="xmi.5"/>
                    </HAREL:ActivityVertex.parent>
                    <HAREL:ActivityVertex.incoming>
                      <HAREL:Flowline xmi.idref="xmi.10"/>
                    </HAREL:ActivityVertex.incoming>
                  </HAREL:Datastore>
                  <HAREL:Activity name="SUBATT1" type="INTERNAL" xmi.id="xmi.7">
                    <HAREL:ActivityVertex.parent>
                      <HAREL:NonBasicActivity xmi.idref="xmi.5"/>
                    </HAREL:ActivityVertex.parent>
                    <HAREL:Activity.Reaction>
                      <HAREL:Expression></HAREL:Expression>
                    </HAREL:Activity.Reaction>
                    <HAREL:NonBasicActivity.SubVertex>
                      <HAREL:Activity         name="CONTROL1"         type="CONTROL"
xmi.id="xmi.8">
                        <HAREL:ActivityVertex.parent>
                          <HAREL:NonBasicActivity xmi.idref="xmi.7"/>
                        </HAREL:ActivityVertex.parent>
                        <HAREL:Activity.Reaction>
                          <HAREL:Expression></HAREL:Expression>
                        </HAREL:Activity.Reaction>
                        <HAREL:ActivityVertex.incoming>
                          <HAREL:Flowline xmi.idref="xmi.11"/>
                        </HAREL:ActivityVertex.incoming>
                      </HAREL:Activity>
                    </HAREL:NonBasicActivity.SubVertex>
```

```
                </HAREL:Activity>
              </HAREL:NonBasicActivity.SubVertex>
            </HAREL:Activity>
            <HAREL:Activity name="EXT1" type="EXTERNAL" xmi.id="xmi.9">
              <HAREL:Activity.activityChart>
                <HAREL:ActivityChart xmi.idref="xmi.3"/>
              </HAREL:Activity.activityChart>
              <HAREL:Activity.Reaction>
                <HAREL:Expression></HAREL:Expression>
              </HAREL:Activity.Reaction>
              <HAREL:ActivityVertex.outgoing>
                <HAREL:Flowline xmi.idref="xmi.10"/>
              </HAREL:ActivityVertex.outgoing>
            </HAREL:Activity>
          </HAREL:ActivityChart.Activities>
          <HAREL:ActivityChart.Flowlines>
            <HAREL:Flowline type="FL_DATA" xmi.id="xmi.10">
              <HAREL:Flowline.activitychart>
                <HAREL:ActivityChart xmi.idref="xmi.3"/>
              </HAREL:Flowline.activitychart>
              <HAREL:Expression>DATA</HAREL:Expression>
              <HAREL:Flowline.source>
                <HAREL:ActivityVertex xmi.idref="xmi.9"/>
              </HAREL:Flowline.source>
              <HAREL:Flowline.target>
                <HAREL:ActivityVertex xmi.idref="xmi.6"/>
              </HAREL:Flowline.target>
            </HAREL:Flowline>
            <HAREL:Flowline type="FL_CONTROL" xmi.id="xmi.11">
              <HAREL:Flowline.activitychart>
                <HAREL:ActivityChart xmi.idref="xmi.3"/>
              </HAREL:Flowline.activitychart>
              <HAREL:Expression>COMMAND</HAREL:Expression>
              <HAREL:Flowline.source>
                <HAREL:ActivityVertex xmi.idref="xmi.5"/>
              </HAREL:Flowline.source>
              <HAREL:Flowline.target>
                <HAREL:ActivityVertex xmi.idref="xmi.8"/>
              </HAREL:Flowline.target>
            </HAREL:Flowline>
          </HAREL:ActivityChart.Flowlines>
        </HAREL:ActivityChart>
      </HAREL:ownedElement>
    </HAREL:Project>
  </HAREL>
 </XMI.content>
</XMI>
```

5.3 Esempio 3

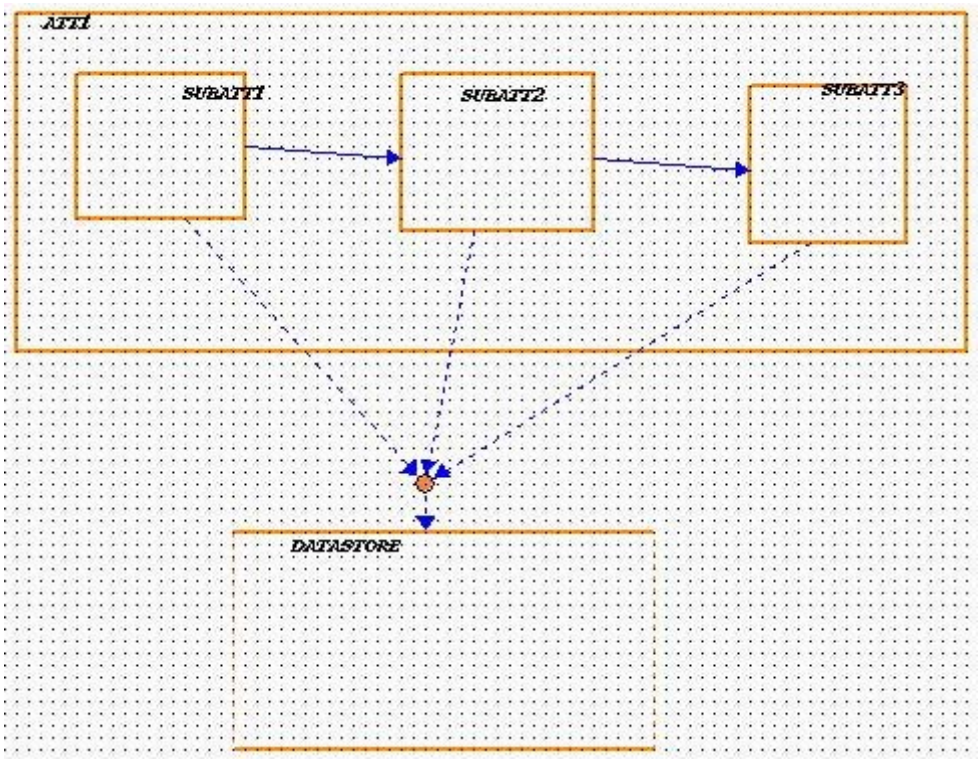

Figura 5 – Esempio 2: esportazione di Activity Charts

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE XMI PUBLIC "Harel" "./harel.dtd">

<XMI xmi.version="1.2">
  <XMI.header>
    <XMI.documentation>
      <XMI.exporter>xmi_exporter</XMI.exporter>
      <XMI.exporterVersion>2.0</XMI.exporterVersion>
    </XMI.documentation>
    <XMI.metamodel xmi.name="Harel.dtd" xmi.version="1.3"/>
  </XMI.header>
  <XMI.content>
    <HAREL xmlns:HAREL='./Harel.dtd'>
      <HAREL:Project name="" xmi.id="xmi.2">
        <HAREL:ownedElement>
          <HAREL:ActivityChart name="ACT_TEST1" xmi.id="xmi.3">
            <HAREL:ActivityChart.context>
              <HAREL:Link xmi.idref="xmi.2"/>
            </HAREL:ActivityChart.context>
            <HAREL:ActivityChart.Activities>
              <HAREL:Activity name="ATT1" type="INTERNAL" xmi.id="xmi.5">
                <HAREL:Activity.activityChart>
                  <HAREL:ActivityChart xmi.idref="xmi.3"/>
                </HAREL:Activity.activityChart>
                <HAREL:Activity.Reaction>
```

```
      <HAREL:Expression></HAREL:Expression>
    </HAREL:Activity.Reaction>
    <HAREL:NonBasicActivity.SubVertex>
      <HAREL:Activity name="SUBATT1" type="INTERNAL" xmi.id="xmi.6">
        <HAREL:ActivityVertex.parent>
          <HAREL:NonBasicActivity xmi.idref="xmi.5"/>
        </HAREL:ActivityVertex.parent>
        <HAREL:Activity.Reaction>
          <HAREL:Expression></HAREL:Expression>
        </HAREL:Activity.Reaction>
        <HAREL:ActivityVertex.outgoing>
          <HAREL:Flowline xmi.idref="xmi.11"/>
          <HAREL:Flowline xmi.idref="xmi.16"/>
        </HAREL:ActivityVertex.outgoing>
      </HAREL:Activity>
      <HAREL:Activity name="SUBATT2" type="INTERNAL" xmi.id="xmi.7">
        <HAREL:ActivityVertex.parent>
          <HAREL:NonBasicActivity xmi.idref="xmi.5"/>
        </HAREL:ActivityVertex.parent>
        <HAREL:Activity.Reaction>
          <HAREL:Expression></HAREL:Expression>
        </HAREL:Activity.Reaction>
        <HAREL:ActivityVertex.outgoing>
          <HAREL:Flowline xmi.idref="xmi.12"/>
          <HAREL:Flowline xmi.idref="xmi.15"/>
        </HAREL:ActivityVertex.outgoing>
        <HAREL:ActivityVertex.incoming>
          <HAREL:Flowline xmi.idref="xmi.11"/>
        </HAREL:ActivityVertex.incoming>
      </HAREL:Activity>
      <HAREL:Activity name="SUBATT3" type="INTERNAL" xmi.id="xmi.8">
        <HAREL:ActivityVertex.parent>
          <HAREL:NonBasicActivity xmi.idref="xmi.5"/>
        </HAREL:ActivityVertex.parent>
        <HAREL:Activity.Reaction>
          <HAREL:Expression></HAREL:Expression>
        </HAREL:Activity.Reaction>
        <HAREL:ActivityVertex.outgoing>
          <HAREL:Flowline xmi.idref="xmi.14"/>
        </HAREL:ActivityVertex.outgoing>
        <HAREL:ActivityVertex.incoming>
          <HAREL:Flowline xmi.idref="xmi.12"/>
        </HAREL:ActivityVertex.incoming>
      </HAREL:Activity>
    </HAREL:NonBasicActivity.SubVertex>
  </HAREL:Activity>
  <HAREL:Datastore name="DATASTORE" xmi.id="xmi.9">
    <HAREL:Activity.activityChart>
      <HAREL:ActivityChart xmi.idref="xmi.3"/>
    </HAREL:Activity.activityChart>
    <HAREL:ActivityVertex.incoming>
      <HAREL:Flowline xmi.idref="xmi.13"/>
    </HAREL:ActivityVertex.incoming>
  </HAREL:Datastore>
  <HAREL:Connector value="" xmi.id="xmi.10">
    <HAREL:Activity.activityChart>
      <HAREL:ActivityChart xmi.idref="xmi.3"/>
    </HAREL:Activity.activityChart>
    <HAREL:Connector.kind xmi.value="act_junction" />
    <HAREL:ActivityVertex.outgoing>
      <HAREL:Flowline xmi.idref="xmi.13"/>
    </HAREL:ActivityVertex.outgoing>
    <HAREL:ActivityVertex.incoming>
```

```
        <HAREL:Flowline xmi.idref="xmi.14"/>
        <HAREL:Flowline xmi.idref="xmi.15"/>
        <HAREL:Flowline xmi.idref="xmi.16"/>
      </HAREL:ActivityVertex.incoming>
    </HAREL:Connector>
  </HAREL:ActivityChart.Activities>
  <HAREL:ActivityChart.Flowlines>
    <HAREL:Flowline type="FL_DATA" xmi.id="xmi.11">
      <HAREL:Flowline.activitychart>
        <HAREL:ActivityChart xmi.idref="xmi.3"/>
      </HAREL:Flowline.activitychart>
      <HAREL:Expression></HAREL:Expression>
      <HAREL:Flowline.source>
        <HAREL:ActivityVertex xmi.idref="xmi.6"/>
      </HAREL:Flowline.source>
      <HAREL:Flowline.target>
        <HAREL:ActivityVertex xmi.idref="xmi.7"/>
      </HAREL:Flowline.target>
    </HAREL:Flowline>
    <HAREL:Flowline type="FL_DATA" xmi.id="xmi.12">
      <HAREL:Flowline.activitychart>
        <HAREL:ActivityChart xmi.idref="xmi.3"/>
      </HAREL:Flowline.activitychart>
      <HAREL:Expression></HAREL:Expression>
      <HAREL:Flowline.source>
        <HAREL:ActivityVertex xmi.idref="xmi.7"/>
      </HAREL:Flowline.source>
      <HAREL:Flowline.target>
        <HAREL:ActivityVertex xmi.idref="xmi.8"/>
      </HAREL:Flowline.target>
    </HAREL:Flowline>
    <HAREL:Flowline type="FL_CONTROL" xmi.id="xmi.13">
      <HAREL:Flowline.activitychart>
        <HAREL:ActivityChart xmi.idref="xmi.3"/>
      </HAREL:Flowline.activitychart>
      <HAREL:Expression></HAREL:Expression>
      <HAREL:Flowline.source>
        <HAREL:ActivityVertex xmi.idref="xmi.10"/>
      </HAREL:Flowline.source>
      <HAREL:Flowline.target>
        <HAREL:ActivityVertex xmi.idref="xmi.9"/>
      </HAREL:Flowline.target>
    </HAREL:Flowline>
    <HAREL:Flowline type="FL_CONTROL" xmi.id="xmi.14">
      <HAREL:Flowline.activitychart>
        <HAREL:ActivityChart xmi.idref="xmi.3"/>
      </HAREL:Flowline.activitychart>
      <HAREL:Expression></HAREL:Expression>
      <HAREL:Flowline.source>
        <HAREL:ActivityVertex xmi.idref="xmi.8"/>
      </HAREL:Flowline.source>
      <HAREL:Flowline.target>
        <HAREL:ActivityVertex xmi.idref="xmi.10"/>
      </HAREL:Flowline.target>
    </HAREL:Flowline>
    <HAREL:Flowline type="FL_CONTROL" xmi.id="xmi.15">
      <HAREL:Flowline.activitychart>
        <HAREL:ActivityChart xmi.idref="xmi.3"/>
      </HAREL:Flowline.activitychart>
      <HAREL:Expression></HAREL:Expression>
      <HAREL:Flowline.source>
        <HAREL:ActivityVertex xmi.idref="xmi.7"/>
      </HAREL:Flowline.source>
```

```
                <HAREL:Flowline.target>
                  <HAREL:ActivityVertex xmi.idref="xmi.10"/>
                </HAREL:Flowline.target>
              </HAREL:Flowline>
              <HAREL:Flowline type="FL_CONTROL" xmi.id="xmi.16">
                <HAREL:Flowline.activitychart>
                  <HAREL:ActivityChart xmi.idref="xmi.3"/>
                </HAREL:Flowline.activitychart>
                <HAREL:Expression></HAREL:Expression>
                <HAREL:Flowline.source>
                  <HAREL:ActivityVertex xmi.idref="xmi.6"/>
                </HAREL:Flowline.source>
                <HAREL:Flowline.target>
                  <HAREL:ActivityVertex xmi.idref="xmi.10"/>
                </HAREL:Flowline.target>
              </HAREL:Flowline>
            </HAREL:ActivityChart.Flowlines>
          </HAREL:ActivityChart>
        </HAREL:ownedElement>
      </HAREL:Project>
    </HAREL>
  </XMI.content>
</XMI>
```

6 Istruzioni per la compilazione e l'esecuzione

6.1 Istruzioni per la compilazione

Unix Solaris Systems

Nella directory in cui risiede il codice sorgente dell'applicazione è presente il file **compile-New**:

```
#!/bin/csh –f
```

- Settare la variabile di ambiente STM_ROOT:
  `setenv STM_ROOT root_name`

`Root_name` è la directory root per StateMate.

```
#setenv STM_ROOT
net/matrix/export/spare/usr.local.solaris8/packages/rep1/STATEMATE33/stmm3.3
setenv STM_ROOT /usr/local/packages/rep1/STATEMATE33/stmm3.3/SOL
```

- Per la compilazione ed il linking eseguire:

```
gcc -o xmi_exporter *.c $STM_ROOT/lib/dataport.o \
$STM_ROOT/lib/libgcc.a $STM_ROOT/lib/x_stubs.o \
-lm -lsocket -lnsl -L /usr/ucblib -lucb –ldl
```

E' possibile usare il file compile (executable c-shell script), fornito insieme ai sorgenti, per eseguire in automatico queste operazioni.

6.2 Istruzioni per l'esecuzione

Per eseguire il file, una volta compilato, è necessario che tutte le variabili di ambiente di cui StateMate ha bisogno siano settate, altrimenti il programma non troverà il file di licenza. Inoltre il linker dinamico ha bisogno di sapere dove sono le librerie incluse. E' possibile utilizzare il file *run* che è stato creato prendendo l'executable c-shell script

run_stmm e sostituendo l'ultima riga che avvia StateMate con:

```
Setenv LD_LIBRARY_PATH /lib\:/usr/lib\:$STM_ROOT/lib\:/usr/ucblib
./xmi_exporter$1$2
```

A questo punto è possibile eseguire il comando:

```
./run workarea[nome_progetto].
```

`Workarea` è la directory dove il progetto da aprire è stato salvato;

`nome_progetto` è opzionale ed è associato al progetto XMI.

Appendice 1 Harel DTD 1.0

Di seguito è riportato il Data Type Definition proposto per la rappresentazione in XML delle macchine a stati di Harel:

```
<!-- Le seguenti due righe includono il DTD dell'XMI contenuto nel file:
./XMI1.2.dtd -->
<!ENTITY % XMI SYSTEM "./XMI1.2.dtd">
%XMI;
<!ELEMENT HAREL ANY>
<!-- _____________________________ HAREL:Project _____________________________ -->
<!ELEMENT HAREL:Project (XMI.extension | HAREL:ownedElement)* >
<!ATTLIST HAREL:Project
name CDATA #IMPLIED
%XMI.element.att;
%XMI.link.att;
>

<!-- _____________________ HAREL:Project.ownedElement _____________________ -->
<!ELEMENT HAREL:ownedElement (XMI.extension | HAREL:StateChart)* >
<!-- _____________________ HAREL:StateMachines _____________________________ -->
<!ELEMENT HAREL:StateMachines (HAREL:StateChart | HAREL:State |
HAREL:CompositeState | HAREL:Connector | HAREL:Note |
HAREL:Transition | HAREL:Event | HAREL:StateVertex |
HAREL:Instance)*
>

<!ATTLIST HAREL:StateMachines
%XMI.element.att;
%XMI.link.att;
>

<!-- _________________________________ HAREL:Link _________________________________-->
<!ELEMENT HAREL:Link (XMI.extension)*>
<!ATTLIST HAREL:Link
%XMI.element.att;
%XMI.link.att;
>

<!-- _____________________________ HAREL:Note _____________________________ -->
<!ELEMENT HAREL:Note.owner (HAREL:Link)* >
<!ELEMENT HAREL:Note (#PCDATA | XMI.extension | HAREL:Note.owner)* >
<!ATTLIST HAREL:Note
%XMI.element.att;
%XMI.link.att;
>

<!-- _____________________ HAREL:StateMachines.StateChart _____________________ -->
<!ELEMENT HAREL:StateChart.context (HAREL:Link)* >
<!ELEMENT HAREL:StateChart.Events (HAREL:Event)* >
<!ELEMENT HAREL:StateChart (XMI.extension | HAREL:StateChart.context |
HAREL:StateChart.States | HAREL:StateChart.Transitions |
HAREL:StateChart.Events | HAREL:Note)*
>

<!ATTLIST HAREL:StateChart
```

```
name CDATA #IMPLIED
%XMI.element.att;
%XMI.link.att;
>

<!-- _____________________ HAREL:StateMachines.Event _____________________ -->
<!-- <!ELEMENT HAREL:Event.transition (HAREL:Transition)* > -->
<!ELEMENT HAREL:Event (XMI.extension | HAREL:StateChart)*>
<!ATTLIST HAREL:Event
name CDATA #IMPLIED
mode (IN | OUT | INOUT | COSTANT) #IMPLIED
%XMI.element.att;
%XMI.link.att;
>

<!-- _____________ HAREL:StateMachines.StateChart.States _____________ -->
<!ELEMENT HAREL:StateChart.States.SubChart (HAREL:Link)* >
<!ELEMENT HAREL:StateChart.States (HAREL:State | HAREL:CompositeState |
HAREL:Connector | HAREL:Instance |
HAREL:StateChart.States.SubChart )*
>

<!-- ___________ HAREL:StateMachines.StateChart.Transitions ___________ -->
<!ELEMENT HAREL:StateChart.Transitions (HAREL:Transition)* >
<!-- ____________________________ HAREL:Expression ____________________________ -->
<!ELEMENT HAREL:Expression.Trigger (HAREL:Event)* >
<!ELEMENT HAREL:Expression.Guard (#PCDATA)* >
<!ELEMENT HAREL:Expression.Effect (HAREL:Event)* >
B.2 Harel DTD 1.0 DTD utilizzati
<!ELEMENT HAREL:Expression (#PCDATA |XMI.extension |
HAREL:Expression.Trigger | HAREL:Expression.Guard |
HAREL:Expression.Effect )*
>

<!-- ________________ HAREL:StateMachines.Transition ________________ -->
<!ELEMENT HAREL:Transition.source (HAREL:StateVertex)* >
<!ELEMENT HAREL:Transition.target (HAREL:StateVertex)* >
<!ELEMENT HAREL:Transition.statechart (HAREL:StateChart)* >
<!ELEMENT HAREL:Transition (XMI.extension | HAREL:Expression |
HAREL:Transition.source | HAREL:Transition.target |
HAREL:Transition.statechart)*
>

<!ATTLIST HAREL:Transition
%XMI.element.att;
%XMI.link.att;
>

<!-- ________________ HAREL:StateMachines.StateVertex ________________ -->
<!ELEMENT HAREL:StateVertex.outgoing (HAREL:Transition)* >
<!ELEMENT HAREL:StateVertex.incoming (HAREL:Transition)* >
<!ELEMENT HAREL:StateVertex.parent (HAREL:CompositeState)* >
<!ELEMENT HAREL:StateVertex (XMI.extension | HAREL:StateVertex.outgoing |
HAREL:StateVertex.incoming | HAREL:StateVertex.parent)*
>

<!ATTLIST HAREL:StateVertex
name CDATA #IMPLIED
%XMI.element.att;
%XMI.link.att;
>

<!-- _____________________ HAREL:StateMachines.State _____________________ -->
```

```
<!ELEMENT HAREL:State.Reaction (HAREL:Expression)* >
<!ELEMENT HAREL:State.stateChart (HAREL:StateChart)* >
<!ELEMENT HAREL:State (XMI.extension | HAREL:StateVertex.outgoing |
HAREL:StateVertex.incoming | HAREL:StateVertex.parent |
B.2 Harel DTD 1.0 DTD utilizzati
HAREL:State.stateChart | HAREL:State.Reaction)*
>

<!ATTLIST HAREL:State
name CDATA #IMPLIED
%XMI.element.att;
%XMI.link.att;
>

<!-- ____________________ HAREL:StateMachines.Instance ____________________ -->
<!ELEMENT HAREL:Instance (XMI.extension | HAREL:StateVertex.outgoing |
HAREL:StateVertex.incoming | HAREL:StateVertex.parent |
HAREL:State.stateChart | HAREL:State.Reaction | HAREL:StateChart)*
>

<!ATTLIST HAREL:Instance
name CDATA #IMPLIED
%XMI.element.att;
%XMI.link.att;
>

<!-- _______________ HAREL:StateMachines.CompositeState _______________ -->
<!ELEMENT HAREL:CompositeState.SubVertex (HAREL:State |
HAREL:CompositeState | HAREL:Connector)*
>

<!ELEMENT HAREL:CompositeState (XMI.extension | HAREL:StateVertex.outgoing |
HAREL:StateVertex.incoming | HAREL:StateVertex.parent |
HAREL:State.stateChart | HAREL:State.Reaction |
HAREL:CompositeState.SubVertex)*
>

<!ATTLIST HAREL:CompositeState
name CDATA #IMPLIED
isConcurrent (false | true) #IMPLIED
%XMI.element.att;
%XMI.link.att;
>

<!-- ____________________ HAREL:StateMachines.Connector ____________________ -->
<!ELEMENT HAREL:Connector.kind EMPTY >

<!ATTLIST HAREL:Connector.kind
xmi.value (default | deepHistory | History | diagram |
selection | junction | joint | termination | condition |
control | composition) #REQUIRED
>

<!ELEMENT HAREL:Connector (XMI.extension | HAREL:StateVertex.incoming
| HAREL:StateVertex.outgoing | HAREL:StateVertex.parent
| HAREL:State.stateChart | HAREL:Connector.kind)*
>

<!ATTLIST HAREL:Connector
value CDATA #IMPLIED
%XMI.element.att;
%XMI.link.att;
>
```