

# Documentazione ModelsTranslator

Simone Antonelli



# Contents

<b>1 ModelsTranslator</b>	<b>5</b>
1.1 Conceptual Analysis . . . . .	5
1.1.1 Package Model . . . . .	6
1.1.1.1 Classe Module . . . . .	6
1.1.1.2 Classe Variable . . . . .	6
Classe Species . . . . .	7
Classe Parameter . . . . .	7
1.1.1.3 Classe Type . . . . .	8
Classe BaseType . . . . .	8
1.1.1.4 Classe Unit . . . . .	8
1.1.1.5 Classe Operation . . . . .	8
Classe BuiltInOp . . . . .	8
Classe UserDefOp . . . . .	8
1.1.1.6 Classe Connection . . . . .	9
Classe DirConnection . . . . .	9
1.1.1.7 Classe Event . . . . .	9
1.1.1.8 Classe Expression . . . . .	10
1.1.2 Package Statements . . . . .	12
1.1.2.1 Stereotipi . . . . .	12
1.1.2.2 Classe Statement . . . . .	13
Classe Return . . . . .	13
Classe Break . . . . .	13
Classe ExpressionStatement . . . . .	13
Classe NestedStatement . . . . .	13
Classe Block . . . . .	14
Classe WhileStatement . . . . .	14
Classe ForStatement . . . . .	14
Classe IfStatement . . . . .	14
Classe WhenStatement . . . . .	15
1.1.2.3 Classe StatementOwner . . . . .	15
1.2 Application Design . . . . .	15
1.2.1 Package Model . . . . .	16
1.2.2 Package Statements . . . . .	17



# Chapter 1

## ModelsTranslator

Il seguente tool, **ModelsTranslator**, si pone l'obiettivo di tradurre i modelli dai principali linguaggi di modellazione, quali Modelica, SBML e PHML, sfruttando un data model costruito appositamente per rappresentare tutte le strutture che è possibile esprimere con l'utilizzo di questi linguaggi. Al momento il tool è in grado di effettuare traduzioni a partire da modelli PHML in modelli Modelica.

### 1.1 Conceptual Analysis

Il data model è stato costruito usando il diagramma della classi UML, ed è diviso in due principali package: uno rappresenta la modellazione degli statement, il secondo rappresenta la modellazione dei vari modelli. Di seguito è mostrata l'immagine che rappresenta l'intero data model:

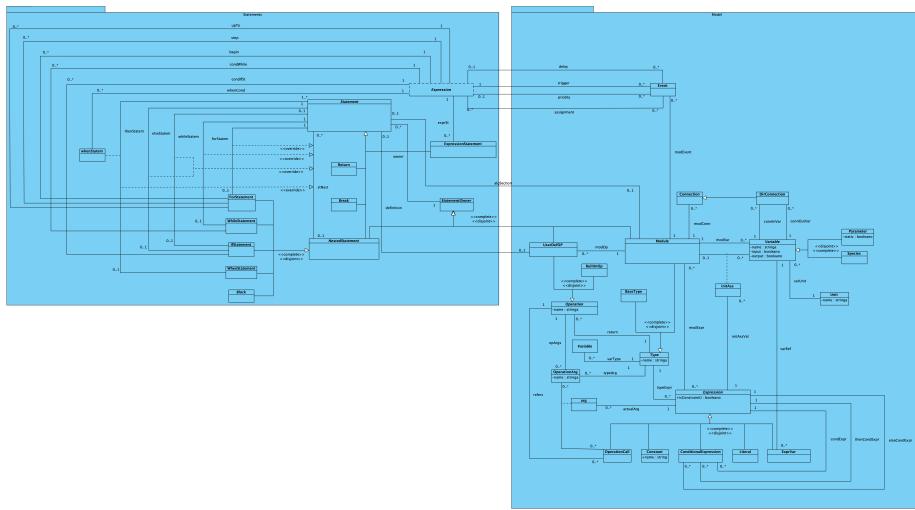


Figure 1.1: *Conceptual Analysis: ModelsTranslator data model.*

Vediamo più nel dettaglio i due package.

### 1.1.1 Package Model

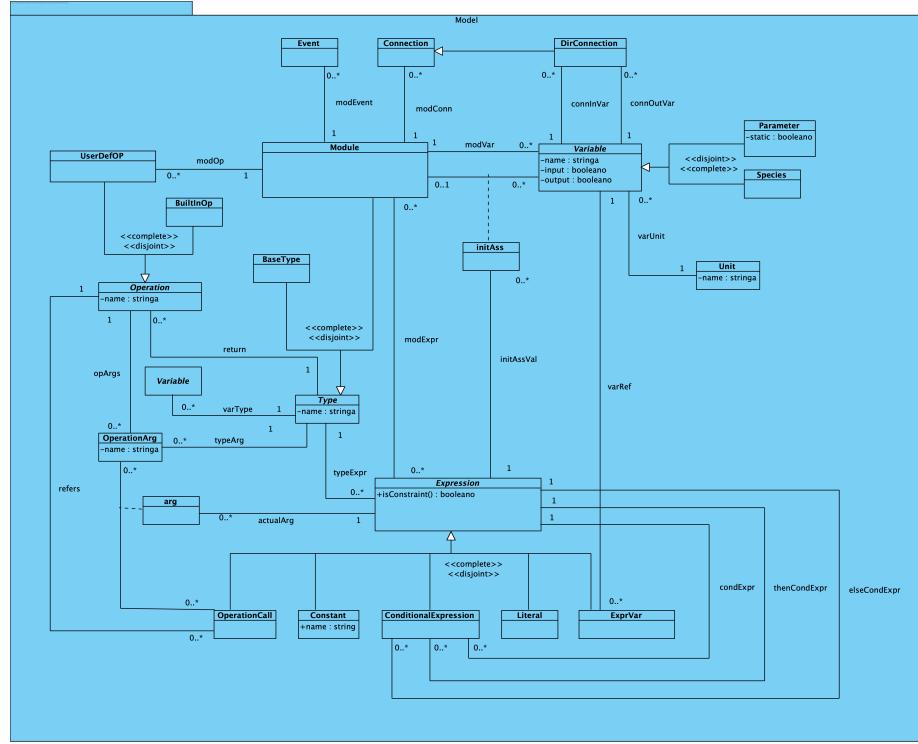


Figure 1.2: Conceptual Analysis: Package Model.

#### 1.1.1.1 Classe Module

La classe *Module* è il core di questo package, e del data model. Un'istanza di questa classe rappresenta il modello vero e proprio.

Le componenti di rilevanza di un *Module* sono le seguenti:

- **variabili**: realizzate tramite l'associazione tra *Module* e *Variable*. Ciascuna istanza rappresenta una variabile del modello;
- **operazioni**: realizzate tramite l'associazione tra *Module* e *UserDefOp*. Ciascuna istanza rappresenta una funzione definita nel modello;
- **connessioni**: realizzate tramite l'associazione tra *Module* e *Connection*. Ciascuna istanza rappresenta una connessione appartente al modello;
- **eventi**: realizzati tramite l'associazione tra *Module* e *Event*. Ciascuna istanza rappresenta un evento definito nel modello;

#### 1.1.1.2 Classe Variable

La classe *Variable* rappresenta delle variabili, ovvero ciò che stabilisce la dinamica di come evolve lo stato di un modello nel tempo. Le componenti di rilevanza di una variabile sono:

- nome: che stabilisce il nome della variabile;
- input: stabilisce se la variabile è di input;
- output: stabilisce se la variabile è di output;
- **tipo**: rappresentato tramite l'associazione tra *Variable* e *Type*. Stabilisce il tipo della variabile;
- **unità**: rappresentata tramite l'associazione tra *Variable* e *Unit*. Stabilisce di che unità è la variabile;
- valore iniziale: rappresentato tramite la classe di associazione *InitAss*. Stabilisce il valore, a tempo di simulazione 0, della variabile, ovvero lo stato di partenza del modello. Il valore viene dato dall'associazione con la classe *Expression*.

Per la seguente classe sono inoltre presenti dei vincoli esterni (espressi in logica del primo ordine), quali:

1. L'assegnamento iniziale di una variabile deve riferirsi ad una del modulo  

$$\forall m, v, e \text{ Module}(m) \wedge initAss(m, v, e) \wedge Expression(e) \wedge Variable(v) \Rightarrow modVar(m, v)$$
2. L'assegnamento iniziale di una variabile deve essere coerente con il tipo della variabile  

$$\forall m, v, e, t \text{ Module}(m) \wedge Variable(v) \wedge Expression(e) \wedge initAss(m, v, e) \wedge Type(t) \wedge varType(t, v) \Rightarrow typeExpr(t, e)$$

La classe *Variable* è una classe astratta, ed è sfruttata per la realizzazione delle classi più generali *Parameter* e *Species*.

**Classe Species** Un'istanza della classe *Species* rappresenta una variabile di stato del modello, ovvero quelle variabili che sono coinvolte in equazioni differenziali ordinarie.

**Classe Parameter** La classe *Parameter*, oltre che alle componenti ereditate dalla classe *Variable*, contiene un'ulteriore componente:

- static: stabilisce se il parametro è statico o meno.

Il concetto di staticità del parametro si riferisce alla simulazione; ovvero se il parametro è statico allora il suo valore non cambia durante la simulazione del modello al quale appartiene, altrimenti il proprio valore è soggetto a variazioni durante la simulazione.

### 1.1.1.3 Classe Type

Un’istanza della classe *Type* rappresenta un data type disponibile nel modello. Le componenti di rilevanza per questa classe sono:

- nome: stabilisce il nome del data type.

La classe *Type* è una classe astratta, utilizzata per la rappresentazione delle classi più specifiche  *BaseType*  e  *Module* .

La classe *Module* rappresenta anche un data type poiché in un modello è possibile definire variabili che rappresentano altri modelli; questo è utile per rappresentare le connessione tra i vari modelli.

**Classe BaseType** Un’istanza della classe  *BaseType*  rappresenta un data type di utilizzo comune. I data type più frequenti nella modellazione, e utilizzati in questo tool sono:

- Integer: per rappresentare un valore intero;
- Real: per rappresentare un valore reale;
- Boolean: per rappresentare un valore booleano;
- String: per rappresentare un valore testuale;

### 1.1.1.4 Classe Unit

Un’istanza della classe *Unit* rappresenta un’unità che può essere associata ad una variabile. L’uso delle unità è utile in quanto consentono di effettuare controlli di consistenza delle unità per le equazioni coinvolte nel modello. Le componenti di rilevanza per questa classe sono:

- nome: stabilisce il nome dell’unità;

### 1.1.1.5 Classe Operation

Un’istanza della classe *Operation* rappresenta un’operazione. Le componenti di rilevanza di questa classe sono:

- nome: stabilisce il nome dell’operazione;
- argomenti: realizzati tramite l’associazione tra *Operation* e *OperationArg*; rappresenta gli argomenti formali dell’operazione;

*Operation* è una classe astratta utilizzata per la realizzazione delle classi più generali *BuiltInOp* e *UserDefOp*.

**Classe BuiltInOp** Le istanze della classe *BuiltInOp* rappresentano le operazioni che vengono fornite di default dai principali linguaggi di modellazione.

**Classe UserDefOp** Le istanze della classe *UserDefOp* rappresentano le operazioni che vengono definite dall’utente in un modello.

### 1.1.1.6 Classe Connection

Un'istanza della classe *Connection* rappresenta una connessione tra due variabili e stabilisce un collegamento tra i modelli. In una connessione può essere importante l'ordine con il quale le variabili sono connesse, e in questo caso la connessione è diretta ed è rappresentata dalla classe *DirConnection*. Le componenti di rilevanza per entrambe le connessioni sono:

- **variabile**: stabilisce la prima variabile coinvolta nella connessione.
- **variabile**: stabilisce la seconda variabile coinvolta nella connessione.

*Nota: nella classe Connection non è importante l'ordine con il quale sono coinvolte le variabili.*

**Classe DirConnection** Nel caso di una connessione diretta è rilevante l'ordine in cui le variabili sono coinvolte nella connessione. Per cui viene fatta una distinzione tra la variabile in input coinvolta nella connessione e la variabile di output coinvolta nella connessione. Questo viene rappresentato attraverso due distinte associazioni tra la classe *DirConnection* e la classe *Variable*.

La seguente classe presenta inoltre i seguenti vincoli esterni:

1. **Variabili coinvolte nella stessa connection, non devono essere la stessa variabile**  

$$\forall m, c, v, v' \text{ } Module(m) \wedge modConn(m, c) \wedge DirConnection(c) \wedge connInVar(c, v) \wedge Variable(v) \wedge connOutVar(c, v') \wedge Variable(v') \Rightarrow v \neq v'$$
2. **Variabili coinvolte nella stessa connection, devono essere dello stesso tipo**  

$$\forall m, c, v, v', t, t' \text{ } Module(m) \wedge modConn(m, c) \wedge DirConnection(c) \wedge connInVar(c, v) \wedge Variable(v) \wedge typeExpr(t, v) \wedge Type(t) \wedge connOutVar(c, v') \wedge Variable(v') \Rightarrow typeExpr(t, v')$$

### 1.1.1.7 Classe Event

Un'istanza della classe *Event* rappresenta un evento. Un evento descrive quando e come avvengono dei cambiamenti di stato istantanei in un modello.

Le componenti di rilevanza di questa classe sono:

- **trigger**: rappresentato dall'omonima associazione tra la classe *Event* e la classe *Expression*. Stabilisce quando l'evento occorre; viene definito attraverso un'espressione booleana, e nel momento in cui questa assume valore di verità "true", l'evento viene attivato;
- **priority**: rappresentata dall'omonima associazione tra la classe *Event* e la classe *Expression*. È possibile che più eventi si verifichino simultaneamente, e per stabilire l'ordine di esecuzione di questi eventi viene utilizzata la priorità.
- **delay**: rappresentato dall'omonima associazione tra la classe *Event* e la classe *Expression*. Stabilisce dopo quanto tempo dall'attivazione dell'evento, quest'ultimo viene eseguito.

- assegnamenti: rappresentati dall'associazione *assignment* tra la classe *Event* e la classe *Expression*. Stabiliscono il nuovo valore per le variabili coinvolte al verificarsi dell'evento.

La classe *Event* presenta inoltre dei vincoli esterni:

1. La trigger condition di un evento deve essere di tipo booleana  
 $\forall m, e, c \text{ Module}(m) \wedge \text{modEvent}(m, e) \wedge \text{Event}(e) \wedge \text{trigger}(e, c) \wedge \text{Expression}(c) \Rightarrow \text{typeExpr}(\text{Boolean}, c)$

2. La priority di un evento deve essere numerico  
 $\forall m, e, p \text{ Module}(m) \wedge \text{modEvent}(m, e) \wedge \text{Event}(e) \wedge \text{priority}(e, p) \wedge \text{Expression}(p) \Rightarrow \text{typeExpr}(\text{Integer}, p) \vee \text{typeExpr}(\text{Real}, p)$

3. Il delay di un evento deve essere numerico  
 $\forall m, e, d \text{ Module}(m) \wedge \text{modEvent}(m, e) \wedge \text{Event}(e) \wedge \text{delay}(e, d) \wedge \text{Expression}(d) \Rightarrow \text{typeExpr}(\text{Integer}, d) \vee \text{typeExpr}(\text{Real}, d)$

#### 1.1.1.8 Classe Expression

Un'istanza della classe *Expression* rappresenta un'espressione. Le espressioni sono utilizzate per assegnare valori a variabili e per stabilire condizioni da soddisfare.

La classe *Expression* è una classe astratta, ed è sfruttata per realizzare le seguenti classi:

- classe *Literal*: ciascuna istanza di questa classe rappresenta le espressioni che consistono di un solo valore letterale;
- classe *Constant*: ciascuna istanza di questa classe rappresenta le espressioni che consistono di un unico simbolo di costante;
- classe *ExprVar*: ciascuna istanza di questa classe rappresenta le espressioni che si riferiscono ad una variabile del modello;
- classe *ConditionalExpression*: ciascuna istanza di questa classe rappresenta le espressioni che, a seconda se la condizione (rappresentata tramite l'associazione *condExpr* tra la classe *ConditionalExpression* ed *Expression*) assume valore di verità "true" o "false", viene valutata in modo differente;
- classe *OperationCall*: ciascuna istanza di questa classe rappresenta le espressioni che si riferiscono a una chiamata a operazione. In questo caso gli argomenti passati sono formali e sono rappresentati dalla classe di associazione *arg*, il cui valore è dato dall'associazione con *Expression*;

La classe *Expression* è anche composta di un'operazione *isConstraint()*, la quale stabilisce se l'espressione considerata è o meno un constraint; il comportamento è descritto di seguito:

- *isConstraint() : bool*

**Pre:**

None

**Post:**

Se il tipo dell'espressione è *Boolean* restituisce "*true*", "*false*" altrimenti.

Inoltre la seguente classe presenta i seguenti vincoli esterni:

1. La condizione di un'espressione condizionale deve essere booleana  
 $\forall e, c, th \ ConditionalExpression(e) \wedge Expression(cond) \wedge condExpr(e, cond) \Rightarrow typeExpr(Boolean, cond)$
2. Il tipo delle espressioni *then* ed *else* di un'espressione condizionale deve essere uguale al tipo dell'espressione condizionale
  - $\forall e, c, th, t \ ConditionalExpression(e) \wedge Expression(th) \wedge thenCondExpr(e, th) \wedge Type(t) \wedge typeExpr(t, e) \Rightarrow typeExpr(t, th)$
  - $\forall e, c, el, t \ ConditionalExpression(e) \wedge Expression(el) \wedge elseCondExpr(e, el) \wedge Type(t) \wedge typeExpr(t, e) \Rightarrow typeExpr(t, th)$
3. Le espressioni coinvolgono solo variabili definite nel modello  
 $\forall e, v, m \ Model(m) \wedge modExpr(m, e) \wedge ExprVar(e) \wedge varRef(e, v) \wedge Variable(v) \Rightarrow modVar(m, v)$
4. Le espressioni coinvolgono solo chiamate a funzioni che sono definite nel modello  
 $\forall e, v, m \ Model(m) \wedge modExpr(m, e) \wedge OperationCall(e) \wedge refers(e, op) \wedge Operation(op) \Rightarrow modOp(m, op)$
5. Il tipo di ritorno di un'operazione deve essere uguale al tipo dell'operazione invocata  
 $\forall op, t, o \ OperationCall(op) \wedge typeExpr(t, op) \wedge Type(t) \wedge refers(op, o) \wedge Operation(o) \Rightarrow return(t, o)$

### 1.1.2 Package Statements

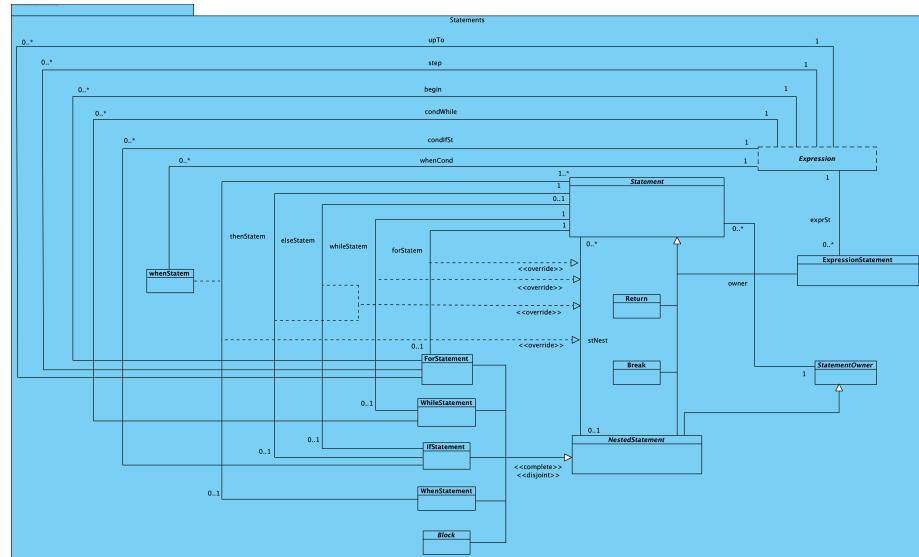


Figure 1.3: Conceptual Analysis: Package Statements.

*Nota: Le classi con i bordi tratteggiati appartengono ad un package differente dal seguente.*

#### 1.1.2.1 Stereotipi

Nel package *Statements* è stato introdotto uno stereotipo il cui comportamento è il seguente:

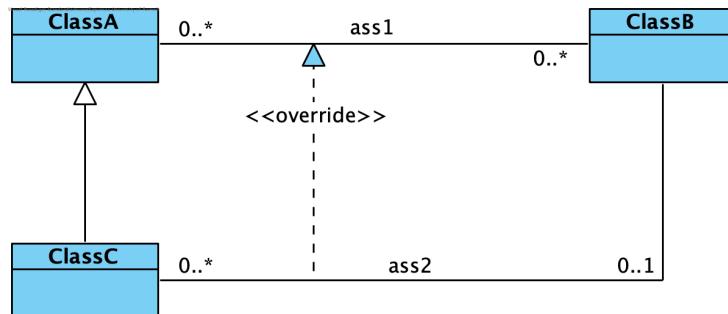
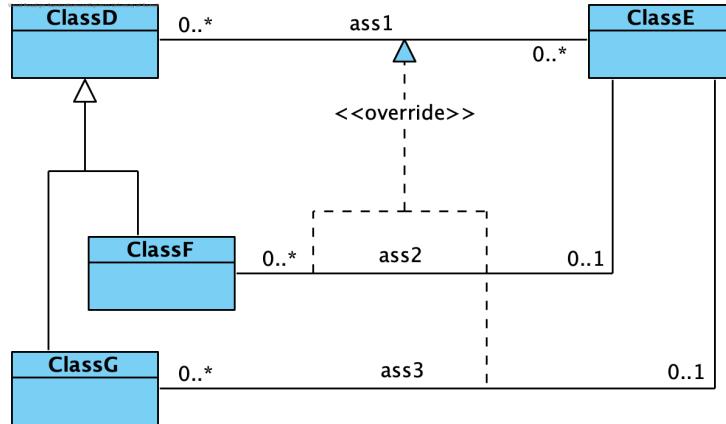


Figure 1.4: Override di una singola associazione.

In presenza del seguente diagramma, l'associazione che realizza quella più generale (ovvero l'associazione indicata dalla freccia) con stereotipo `<>override<>`, prende il posto dell'associazione più generale. Le molteplicità dell'associazione che effettua la realizzazione possono essere più specifiche delle molteplicità dell'associazione generale. Nel caso di due associazioni lo stereotipo si comporta nel seguente modo:

Figure 1.5: *Override di due associazioni.*

In presenza del seguente diagramma, si ha che le associazioni che realizzano quella più generale con stereotipo `<<override>>` andranno a sostituire l'associazione più generale. Inoltre entrambe le associazioni saranno realizzate in modo disgiunto e completo. Le molteplicità delle associazioni che effettuano la realizzazione possono essere più specifiche delle molteplicità dell'associazione generale.

### 1.1.2.2 Classe Statement

Proprio come *Module* è la classe principale del package precedente, la classe *Statement* è il core di questo package, e, come è intuibile dal nome, ogni sua istanza rappresenta uno statement. Le componenti di rilevanza di questa classe sono:

- proprietario dello statement: realizzata dall'associazione tra la classe *Statement* e la classe *StatementOwner*. Rappresenta quindi a quale entità lo statement appartiene;

La classe *Statement* è una classe astratta utilizzata per la rappresentazione delle classi *Break*, *Return*, *ExpressionStatement* e *NestedStatement*.

**Classe Return** Ciascun istanza della classe *Return* rappresenta un return statement, ovvero uno statement che consiste della sola operazione di return.

**Classe Break** Come per la classe *Return*, ogni istanza della classe *Break* rappresenta uno statement che consiste della sola operazione di break.

**Classe ExpressionStatement** Un'istanza della classe *ExpressionStatement* rappresenta uno statement che consiste di una sola espressione. Le componenti di rilevanza di questa classe sono;

- **espressione**: rappresentata dall'associazione tra la classe *ExpressionStatement* e la classe *Expression*. Stabilisce da quale espressione sarà composto lo statement;

**Classe NestedStatement** Un'istanza della classe *NestedStatement* rappresenta uno statement che è composto da più statement dei quali è importante preservarne l'ordine. Le componenti di rilevanza di questa classe sono:

- **statements**: rappresentati dall'associazione tra la classe *NestedStatement* e la classe *Statement*. Stabilisce gli statement di cui è composto il *NestedStatement*;

La classe *NestedStatement* è una classe astratta data la possibilità di rappresentare diverse tipologie di statement che sono a loro volta composti da più statement; viene utilizzata per la realizzazione delle classi più specifiche *Block*, *ForStatement*, *WhileStatement*, *IfStatement* e *WhenStatement*.

**Classe Block** Ciascun istanza della classe *Block* rappresenta semplicemente uno statement che a sua volta consiste di più statement. Anche in questo caso è importante preservare l'ordine degli statement di cui è composto il blocco.

**Classe WhileStatement** Ogni istanza della classe *WhileStatement* rappresenta il costrutto iterativo while; in questo caso il numero degli statement di cui è composto il *WhileStatement* è più specifico del numero di statement di un *Nested<statement>* e per questo motivo viene applicato l' $\langle\langle override\rangle\rangle$  all'associazione della classe più generale. Le componenti di rilevanza di questa classe sono:

- condizione: realizzata dall'associazione tra la classe *WhileStatement* e la classe *Expression*. Stabilisce la condizione del costrutto while, ovvero quando il while deve essere eseguito;
- statement: realizzato dall'associazione tra la classe *WhileStatement* e la classe *Statement*. Stabilisce l'**unico** statement del quale è composto il costrutto while.

La seguente classe presenta inoltre i seguenti vincoli esterni:

1. La condizione di uno statement while deve essere booleana  
 $\forall \text{WhileStatement}(s) \wedge \text{Expression}(e) \wedge \text{condWhile}(s, e) \Rightarrow \text{typeExpr}(\text{Boolean}, e)$

**Classe ForStatement** Ciascuna istanza della classe *ForStatement* rappresenta il costrutto iterativo for; anche in questo il numero di statement di cui è composto è più specifico del numero di statement di cui è composta la classe più generale *NestedStatement*, e quindi all'associazione più generale viene applicato l' $\langle\langle override\rangle\rangle$ . Le componenti di rilevanza di questa classe sono:

- index start: realizzato dall'associazione *begin* tra la classe *ForStatement* e la classe *Expression*. Rappresenta l'inizializzazione dell'indice del costrutto for;
- index step: realizzato dall'associazione *step* tra la classe *ForStatement* e la classe *Expression*. Rappresenta di quanto l'indice deve essere aumentato ad ogni iterazione del costrutto for;
- index up to: realizzato dall'associazione *upTo* tra la classe *ForStatement* e la classe *Expression*. Rappresenta il valore che l'indice deve raggiungere per far terminare il ciclo for;
- statement: realizzato dall'associazione tra la classe *ForStatement* e la classe *Statement*. Stabilisce l'**unico** statement del quale è composto il costrutto for.

**Classe IfStatement** Ciascun istanza della classe *IfStatement* rappresenta il costrutto condizionale if-then-else; per poter rappresentare sia il ramo then che il ramo else del costrutto si ricorre all'utilizzo dello stereotipo  $\langle\langle override\rangle\rangle$  che coinvolge due associazioni, entrambe con molteplicità ristretta rispetto all'associazione più generale. Le componenti di rilevanza di questa classe sono:

- condizione: realizzata tramite l'associazione tra la classe *IfStatement* e la classe *Expression*. Stabilisce la condizione del costrutto if, ovvero la condizione per cui il ramo then del costrutto deve essere eseguito;
- then statement: realizzato tramite l'associazione *thenStatement* tra la classe *IfStatement* e la classe *Statement*. Stabilisce l'**unico** statement del quale è composto il ramo then del costrutto if;

- else statement: realizzato tramite l'associazione *elseStatement* tra la classe *IfStatement* e la classe *Statement*. Stabilisce l'**unico** statement del quale è composto il ramo else del costrutto if. Questa componente è opzionale poiché il costrutto if può essere privo del ramo else;

Inoltre la classe *IfStatement* contiene i seguenti vincoli esterni:

1. La condizione di uno statement if deve essere booleana

$$\forall IfStatement(s) \wedge Expression(e) \wedge condIfSt(s, e) \Rightarrow typeExpr(Boolean, e)$$

**Classe WhenStatement** Ciascuna istanza della classe *WhenStatement* rappresenta il costrutto condizionale when; anche in questo scenario il numero di statement del costrutto when è più specifico del numero di statement della classe più generale *NestedStatement*, per cui all'associazione più generale è stato applicato lo stereotipo `\langle override \rangle`. Le componenti di rilevanza di questa classe sono:

- (condizione, statement): realizzate dalla classe associazione *whenStatement* tra la classe *WhenStatement* e la classe *Statement*, dove la condizione è data dall'associazione tra *whenStatement* e la classe *Expression*. Stabilisce gli statement di cui è composto il costrutto when, ognuno dei quali è associato a una condizione che determina quando lo statement deve essere eseguito;

La classe *WhenStatement* presenta i seguenti vincoli esterni:

1. La condizione di uno statement when deve essere booleana

$$\forall WhenStatement(w) \wedge Statement(s) \wedge WhenStatement(w, s, e) \wedge Expression(e) \Rightarrow typeExpr(Boolean, e)$$

### 1.1.2.3 Classe StatementOwner

Ogni istanza della classe *StatementOwner* rappresenta il proprietario di uno *Statement*. Ogni *Statement* ha un **unico** proprietario che può corrispondere a classi diverse. Per questo motivo la classe *StatementOwner* è una classe astratta utilizzata per rappresentare il caso in cui uno *Statement* ha come proprietario un *Module*, oppure quando ha come proprietario un *operazione definita dall'utente* (in questo caso definisce il corpo dell'operazione), oppure quando ha come proprietario un *NestedStatement* (in questo caso lo statement farà parte degli statement che si trovano all'interno di un *NestedStatement*).

## 1.2 Application Design

Di seguito viene presentato il diagramma UML di design, costruito a partire dal diagramma di analisi precedentemente descritto. Durante questa fase per ogni classe sono state introdotte le operazioni che nella fase di implementazione sono state realizzate.

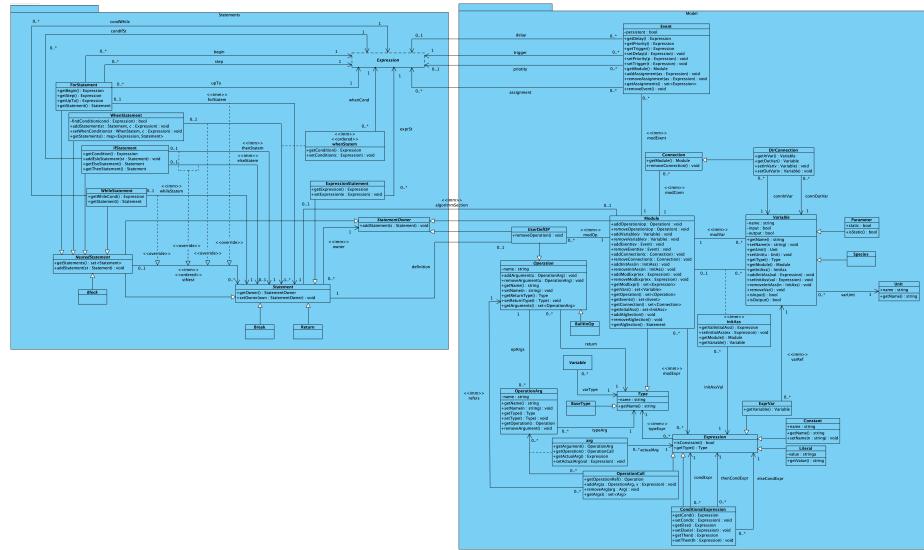


Figure 1.6: Application Design: ModelsTranslator data model.

### 1.2.1 Package Model

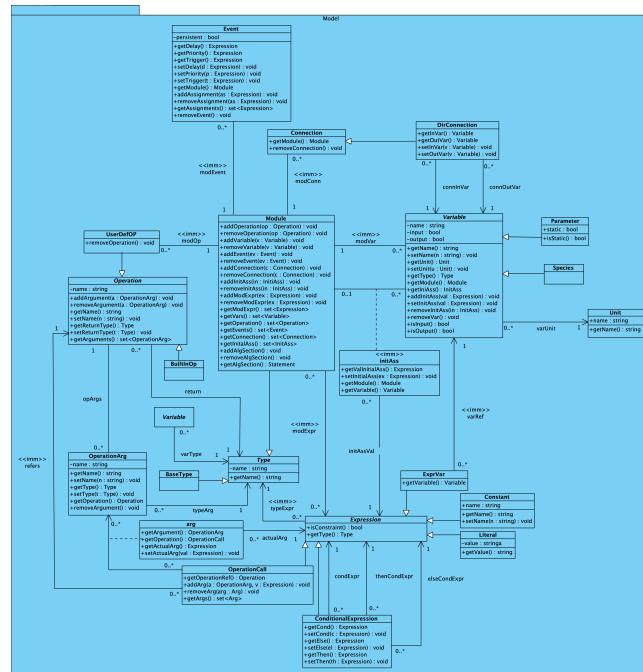


Figure 1.7: Application Design: Package Model.

### 1.2.2 Package Statements

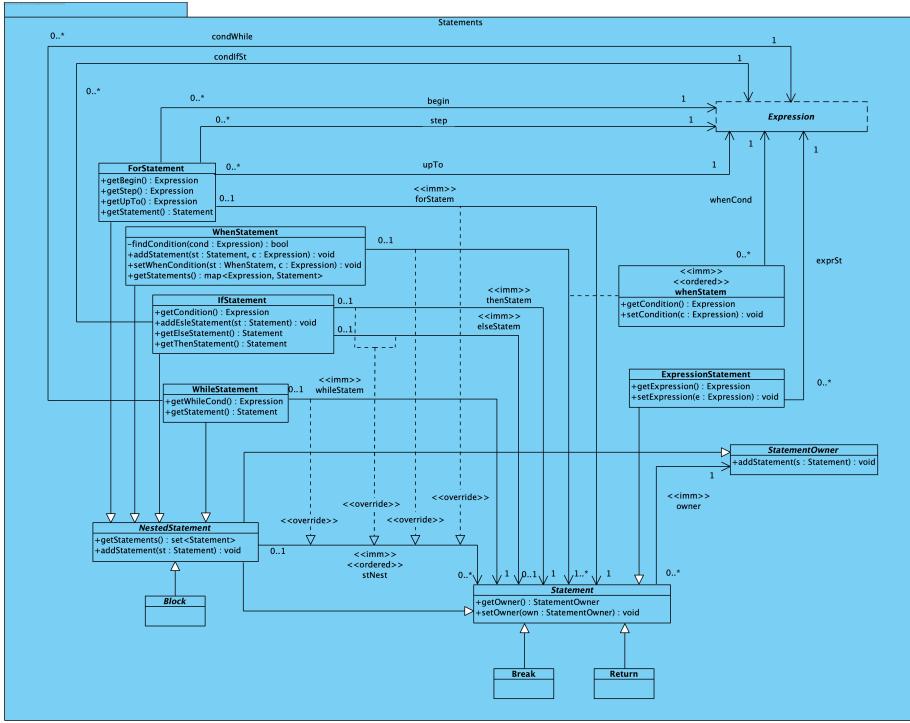


Figure 1.8: Application Design: Package Statements.