Intesa GIS / WG 01 **1n 1010_1**

INTESA STATO REGIONI ENTI - LOCALI
SISTEMI INFORMATIVI TERRITORIALI

COMITATO TECNICO DI COORDINAMENTO

SPECIFICHE PER LA REALIZZAZIONE DEI DATA BASE TOPOGRAFICI DI INTERESSE GENERALE

TITOLO:

IL MODELLO CONCETTUALE
GeoUML
SPECIFICA FORMALE IN UML

Data di emissione:	7 aprile 2004
Versione.sottoversione:	2.1
Tipo di documento:	Versione definitiva per la sperimentazione
Emesso da:	Intesa GIS / WG 01
Riferimenti:	1n1010_2
Nome del file:	1n1010_1.pdf
URL:	http://www.intesagis.it
Proprietà intellettuale e limitazioni d'uso:	La proprietà intellettuale è condivisa dagli Enti partecipanti all'IntesaGIS. Il contenuto può essere liberamente utilizzato e riprodotto, nell'ambito degli scopi previsti dall'IntesaGIS e delle finalità del documento, con obbligo di citazione della fonte.

NOTA: INTESAGIS STA PER INTESA STATO REGIONI ENTI LOCALI SUI SISTEMI INFORMATIVI TERRITORIALI

Intesa GIS / WG 01 **1n 1010_1**

Abstract:

Documento di riferimento per la specifica formale in UML
Il modello concettuale GeoUML

Redazione:	Alberto Belussi, Mauro Negri, Giuseppe Pelagatti (responsabile)
Note	Lavoro svolto dal Dipartimento di Elettronica e Informazione (DEI) del Politecnico di Milano su contratto di ricerca finanziato dal progetto IntesaGIS
Intesa GIS /WG 01: Gruppo di lavoro Specifiche dei DB Topografici	Gennaro Afeltra, Alberto Belussi, Flavio Bernabino, Lorenzo Bottai, Manuela Corongiu, Stefania Crotta, Lino Di Rienzo, Dario Dominico, Marco D'Orazi, Roberto Gaspani, Gabriele Garnero, Franco Guzzetti, Federica Liguori, Mauro Negri, Mauro Nordio, Stefano Olivucci, Sergio Panella, Giuseppe Pelagatti, David Remotti, Mauro Rossi (coordinamento), Umberto Sassoli, Antonio Trebeschi, Mauro Vasone, Antonio Zampieri
Supporto Scientifico DB Spaziali	Giuseppe Pelagatti (PoliMI)
Esperti incaricati della revisione dei documenti	Sergio Dequal (PoliTo), Mario Fondelli (Iuav), Riccardo Galetto (UniPv), Luciano Surace (IIM)
La struttura dell'IntesaGIS	*Il coordinamento ed indirizzo complessivo sulle attività dell'IntesaGIS è svolto dal Comitato Tecnico di Coordinamento composto dai rappresentanti dello Stato (organi cartografici), delle Regioni e degli Enti Locali*
Comitato Tecnico di Coordinamento	Carlo Cannafoglia - presidente (Agenzia Territorio), Maurizio De Gennaro e Aldo Marolla - segreteria CTC (Reg.Veneto), Gianfranco Amadio (IGM), Vincenza Buccino (Reg. Basilicata), Claudio Cattena (Reg.Lazio), Maria Donatella Borsellino (Reg. Sicilia), Elettra Cappadozzi (CNIPA), Raffaele Caputo (ANCI), Carlo Dardengo (IIM), Mario Di Massa (CONFSERVIZI), Roberto Gavaruzzi (Reg. Emilia Romagna), Roberto Laffi (Reg. Lombardia), Angelo Lisi (APAT). Domenico Longhi (Reg. Abruzzo), Enrico Nardelli (UNICEM), Sebastiano Rao (Reg. Piemonte), Giovanni Tomei (UPI), Giampaolo Turco (CIGA), Marcello Vitiello (Reg. Molise).
Struttura di coordinamento e verifica DB Topografici per il CTC	Mario Desideri (Reg. Toscana) e Gianfranco Amadio (IGM) - responsabili, Giampaolo Artioli (Reg. Emilia-Romagna), Maria Donatella Borsellino (Reg. Sicilia), Elettra Cappadozzi (CNIPA),, Stefania Crotta (Reg. Lombardia), Sergio Farruggia (Comune Genova) , Roberto Gaspani (Comune Bergamo), Antonio Venditti (Min. Ambiente) Marcello Vitiello (Reg. Molise)

Parole chiave: Specifiche di contenuto, documento di riferimento, versione definitiva per sperimentazione, modello concettuale, GeoUML, UML

PREMESSA

Con la pubblicazione di questa versione dei documenti di "Specifica per la realizzazione dei Data Base Topografici di interesse generale" termina la fase di definizione preliminare dei contenuti e si avvia la sperimentazione attraverso alcune applicazioni pilota anche su scala estesa, della durata indicativa di un biennio. Una modalità del tutto simile a quanto avviene per gli standard Europei di settore, che prevedono una fase di validazione biennale (ENV).
Nel corso della sperimentazione si provvederà a completare i documenti e le parti ancora mancanti e a sviluppare gli approfondimenti già previsti.
Completata questa ulteriore fase le Specifiche verranno proposte alla Conferenza Stato Regioni Enti locali per la loro approvazione così come stabilito dall'Intesa sui sistemi informativi geografici.

Una tale sperimentazione risulta quanto mai necessaria a fronte della complessità derivante dalla convergenza di molteplici aspetti e dall'innovazione tecnologica sottintesa, ed ha come scopo primario la verifica dei seguenti aspetti:

- **Le modalità di effettiva realizzazione della Base Dati Topografica**. Con quali parametri di qualità a fronte di quali tempi e costi. Una verifica complessiva e di dettaglio sia per una fornitura di primo impianto, sia per la derivazione, con o senza aggiornamento fotogrammterico, da CTR numerica esistente presso gli Enti. La sperimentazione deve permettere di sottoporre a controllo ogni suo aspetto in un contesto di una casistica estesa e non solo più prototipale e deve coinvolgere in questa fase l'esperienza di tutti gli operatori del settore, dagli utenti alle ditte fornitrici di cartografia e GIS;

- **la fruibilità della Base dati Topografica.** Il grado di adeguatezza a fronte dei tanti e tanto dissimili utilizzi con cui deve integrarsi, intendendo con questo sia la fruibilità diretta dei suoi contenuti, ma e soprattutto, la sua adeguatezza ad essere integrata nelle diverse basi dati delle applicazioni di settore.
  Quale sia la sua potenzialità effettiva a costituire una prima base condivisa, che possa esser anche il presupposto per una più vasta opera di integrazione e condivisione tra basi dati. Una fruibilità diretta quindi che si innesti nel flusso informativo di un Ente, garantendosi in tal modo l'aggiornamento dei suoi dati in tempo reale, ed una fruibilità tematica e applicativa, come nucleo condiviso e condivisibile di tutte le informazioni territoriali;

- **L'effettivo grado di interoperabilità**. La sperimentazione di quale grado di interoperabilità si può instaurare tra i diversi Enti od Uffici che aderiscono all'IntesaGIS, a verifica di uno dei presupposti fondanti di tutto il progetto. Con quali modalità, quali regole e con quale efficienza. Quale la reale suddivisione e distribuzione tra gli Enti e nel territorio, nell'ambito del contesto operativo nazionale;

- **la derivabilità del DB25** in tutti i casi reali e soprattutto cercando di minimizzare i requisiti necessari per tale derivazione;

- **la sua integrazione nel Sistema Informativo** di un Ente o di un Ufficio. Quali problematiche e quali soluzioni ottimali nella progettazione e la realizzazione del proprio Database, del proprio ambiente di elaborazione spaziale e di gestione dell'informazione territoriale (GIS). Quali problemi e quali soluzioni per una condivisione in rete efficiente e con quali tecnologie.

Risulta evidente come i punti precedentemente elencati si intreccino e si intersechino in una sperimentazione complessiva rivolta tutti gli aspetti.

Per garantire la massima ricaduta, nella fase di revisione dei documenti, dei risultati conseguiti dalle sperimentazioni, risulterà fondamentale un loro coordinamento con la direzione del progetto IntesaGIS, cui potranno rivolgersi anche per ogni approfondimento delle Specifiche stesse.

Un ulteriore aspetto che dovrà esser affrontato in questa fase riguarda l'aggiornamento professionale connesso alla produzione e utilizzo dei DB topografici . Una tale competenza, sia degli utenti sia dei fornitori di dati, è tutt'altro che secondaria e risulterà decisiva per un reale successo di tutto il progetto.

Come meglio specificato nel documento "**Le Specifiche per la realizzazione dei Database Topografici di interesse generale - lo stato dell'arte ed alcune proposte per una prosecuzione**", le Specifiche sin qui prodotte rappresentano un primo nucleo che richiede di essere ulteriormente integrato da approfondimenti relativi all'informazione catastale, alla codifica delle Entità e degli attributi, ad una presentazione cartografica dinamica, adeguata alle nuove tecnologie di rete, alla derivazione della presentazione a scale di sintesi oltre che del DB25, solo per citare i più importanti.

Non meno importante sarà stabilire quale precisione dei dati sarà necessaria a fronte dell'imminente impiego del GPS associato ad una rete UMTS e quale struttura dati. Quale precisione a fronte delle elaborazioni necessarie alla gestione del dissesto idrogeologico, o quale densità informativa e quale aggiornamento sono richiesti da una efficiente gestione del Servizio Nazionale di Protezione Civile.

Occorre inoltre approfondire quali frontiere stabilire per la terza dimensione a fronte delle nuove tecnologie, quali ad esempio quella del Lidar, e delle funzionalità di elaborazione delle stesse e delle emergenti esigenze.

Una caratteristica del progetto, non meno importante e quanto mai attuale, riguarda la sua naturale convergenza in quello più complessivo che sta nascendo per una Infrastruttura Nazionale di gestione dei Dati Spaziali, NSDI integrata a livello europeo, ESDI: il progetto INSPIRE di cui il progetto IntesaGIS può costituire la modalità di realizzazione del nucleo nazionale di base posizionato tra i più evoluti.

Uno sforzo coordinato in questa direzione permetterà a tutto il contesto nazionale di collocarsi adeguatamente in quello europeo e di far fronte in modo efficiente alle nuove emergenti e pressanti richieste nel campo dell'elaborazione dei dati territoriali, dotandosi di quella che ormai risulta esser una infrastruttura fondamentale per la gestione e lo sviluppo del territorio.

Si è giunti alla fine di questa fase del lavoro e alla soddisfazione di un obiettivo raggiunto si unisce la consapevolezza delle difficoltà che abbiamo ancora davanti, degli ostacoli da superare per migliorare gli elaborati e completare le parti mancanti e soprattutto per farle diventare patrimonio comune e base di un programma nazionale di produzione dell'informazione geografica.

Ci preme infine ringraziare tutti coloro che hanno collaborato per raggiungere questi risultati: in primo luogo il Gruppo di lavoro e i diversi redattori dei documenti; i colleghi del Gruppo di coordinamento DB topografici, gli esperti di riferimento rappresentativi della Comunità scientifica nazionale, tutti i tecnici, professionisti ed utenti degli enti pubblici, dei centri di ricerca, delle imprese ed associazioni che hanno animato gli incontri ed i confronti finora realizzati e che non mancheranno di partecipare al prossimo Convegno di Venezia di presentazione dei risultati.

Carlo Cannafoglia
Mario Desideri
Gianfranco Amadio

INDICE

1 Introduzione

Questo documento contiene la definizione del modello concettuale per la specifica del contenuto del database topografico dell'Intesa Stato-Regioni-EntiLocali, chiamato nel seguito GeoUML, e le regole di corrispondenza tra GeoUML e il linguaggio UML. (Nota Bene: GeoUML non coincide con ciò che è chiamato "Conceptual Model" in ISO/TC 211, ma piuttosto con il "General Feature Model" presentato nello stesso standard).

GeoUML costituisce una specializzazione degli standard definiti dall'ISO/TC 211, seguendo l'impostazione proposta dagli stessi standard, che sono molto generali. Esso si basa su una serie di *costrutti di base,* molto vicini a quelli elementari del General Feature Model di ISO, per garantire la generalità e flessibilità del modello e vi costruisce sopra un insieme di *costrutti derivati*, più potenti ma meno generali, che costituiscono il principale valore aggiunto di questo modello rispetto all'adozione diretta degli standard senza specializzazione.

I costrutti derivati possono essere considerati delle forme abbreviate per indicare porzioni di schema che potrebbero essere espresse con i costrutti di base; il loro uso però è fondamentale in quanto:

- semplifica la scrittura dello schema e lo rende molto più leggibile;
- predefinisce una strutturazione ben organizzata dello schema (anche ai fini della implementabilità).

(i due tipi di costrutti possono essere assimilati ai mattoni e ai prefabbricati, purché questi siano fatti a loro volta di mattoni)

Inoltre, l'impostazione adottata consente di prevedere l'arricchimento ulteriore del modello tramite l'aggiunta di ulteriori costrutti derivati, se lo si riterrà opportuno a fronte dell'evoluzione delle specifiche da un lato e della comprensione delle strutture logiche implementative dall'altro.

1.1 Formalizzazione e compatibilità

La compatibilità di GeoUML con gli standard ISO/TC211 si basa sulla aderenza alle regole definite nel documento ISO DIS 19109 "Rules for application schema", in particolare sulla traduzione in UML dei costrutti del modello stesso, e sull'impiego, per gli attributi geometrici, di una specializzazione delle classi definite nel documento ISO DIS 19107 "Spatial schema".

Dato che UML è un linguaggio formale di specifica, la traduzione in UML del GeoUML costituisce anche una specifica formale dello stesso. Lo schema complessivo di traduzione è rappresentato in Figura 1.

I costrutti di base di GeoUML sono estremamente vicini a quelli del linguaggio UML, per cui la loro traduzione è banale. In particolare, tra i costrutti di base è incluso il linguaggio OCL (object constraint language), cioè la parte di UML dedicata all'espressione di vincoli (che risultano quindi espressi direttamente in UML e non abbisognano di traduzione). Utilizzando OCL è possibile esprimere vincoli molto potenti ed articolati, ma la leggibilità dello schema risulta ridotta e l'implementazione di tali vincoli risulta assai complessa.

La traduzione dei costrutti derivati, che si trasformano in espressioni complesse sui costrutti di base, in particolare espressioni OCL, è molto più difficile della traduzione dei costrutti di base in UML. Queste espressioni complesse costituiscono dei "prefabbricati" inclusi in GeoUML in modo da evitare all'utilizzatore di dover scrivere le corrispondenti espressioni e da permettere di studiare la loro implementazione a priori una volta per tutte. Per l'utilizzatore meno esperto i costrutti derivati permettono di applicare concetti intuitivamente più semplici, senza la necessità di comprendere la loro definizione formale.

Tra i costrutti derivati, quelli più articolati sono sicuramente i vincoli topologici, che permettono di definire una strutturazione della geometria secondo proprietà topologiche. La complessità della loro formalizzazione è dovuta a diversi aspetti:

- la complessità e ricchezza intrinseca delle relazioni spaziali
- la dipendenza della definizione delle relazioni spaziali dai tipi di oggetti geometrici ai quali si applicano
- la generalità della nozione, peraltro fondamentale, di "complesso geometrico" (GM_Complex) nello Spatial Schema
- la compresenza, in un vincolo, delle citate complicazioni di natura geometrica, con quelle derivanti dalla logica, in particolare la quantificazione esistenziale e universale.

Come già detto, l'utilizzatore inesperto può utilizzare i costrutti derivati senza bisogno di comprendere la loro definizione formale (almeno nei casi in cui l'interpretazione intuitiva non induce ambiguità); tuttavia, lo scopo di questo documento è quello di fornire la specifica di GeoUML e non una guida intuitiva all'uso del modello, per la quale si rimanda al documento [1n1010_2] "Il modello concettuale GeoUML – Inquadramento generale e introduzione all'uso". Pertanto, la descrizione intuitiva fornita in questo documento è secondaria e limitata, anche se vengono forniti alcuni esempi; si consiglia pertanto di leggere questo documento dopo aver visto il documento [1n1010_2].

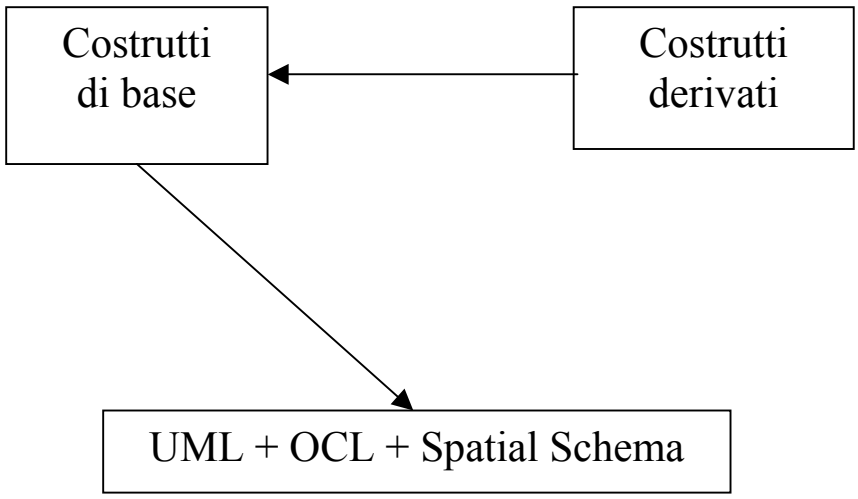


Figura 1 Traduzione diretta e indiretta dei costrutti di GeoUML in UML

1.2 Struttura del documento

Il capitolo 2 è dedicato alla definizione dei costrutti di base. Per ogni costrutto vengono fornite:

- la definizione ,
- la rappresentazione testuale,
- la corrispondente rappresentazione grafica,
- la traduzione in UML, che costituisce anche la formalizzazione del modello,
- un esempio di applicazione e commenti.

Il capitolo 3 tratta in maniera analoga i costrutti derivati, ma la struttura della presentazione è in alcuni punti differenziata per permettere una trattazione più articolata. Le sezioni più lunghe e articolate dell'intero documento sono la 3.4, 3.5 e 3.6, che trattano i vincoli di natura topologica.

La comprensione del documento richiede di conoscere il linguaggio OCL. Per facilitarne la lettura, l'appendice 1 contiene una presentazione sintetica delle caratteristiche del linguaggio OCL più usate in questo documento.

1.3 Riferimenti

I documenti di riferimento sono:

Progetto Intesa Stato-Regioni-Enti locali

- [1n1010_2] G. Pelagatti "Il modello concettuale GeoUML – Inquadramento generale e introduzione".

Standard ISO/TC 211:

- ISO DIS 19101 "Reference model"
- ISO DTS 19103 "Conceptual schema language"
- ISO DIS 19107 "Spatial schema"
- ISO DIS 19109 "Rules for application schema"

Object Management Group (OMG):

- OMG Unified Modeling Language Specification (version 1.3)

Letteratura scientifica sulle relazioni topologiche:

- [Clementini93] E. Clementini, P. Di Felice, and P. van Oosterom., A Small Set of Formal Topological Relationships Suitable for End-User Interaction. In Proceedings 3nd Symposium on Spatial Databases, 1993, pages 277-295.

1.4 Versioni precedenti del modello concettuale

In precedenza, per la specifica del database topografico dell'Intesa (documento 1n1004) erano stati utilizzati i concetti definiti nel documento 1n1002. Tali concetti non erano completi e la loro compatibilità con gli standard ISO/TC 211 non era stata verificata. Il presente GeoUML sostituisce integralmente il documento 1n1002.

Le principali differenze rispetto al documento 1n1002 riguardano le seguenti aree:

- adozione, oltre alla notazione testuale, di una notazione grafica simile a quella dell'UML
- adozione rigorosa delle classi geometriche dello Spatial Schema di ISO TC 211
- strutturazione a due livelli (costrutti base e costrutti derivati)
- definizione di una serie di vincoli standard che generalizzano i vincoli maggiormente usati nella prima esperienza di definizione del DB topografico (documento 1n1004)
- completamenti vari: nozione di chiave primaria, attributi multivalore, attributi a tratti dinamici

2 Costrutti di base

In questo capitolo sono definiti i costrutti di base del modello concettuale GeoUML indicando per ognuno di essi:

- la definizione,
- la rappresentazione testuale,
- la corrispondente rappresentazione grafica,
- la traduzione in UML, che costituisce anche la specifica formale del costrutto,

Inoltre, in molti casi è inclusa una discussione intuitiva e un esempio di applicazione.

Dato che i costrutti di base sono molto simili a quelli dell'UML, la notazione grafica adottata è molto simile a quella dell'UML (in sostanza, conoscendo UML si possono leggere i costrutti di base di GeoUML senza fatica e viceversa).

Tuttavia, dato che in UML non è definita una forma testuale, la forma testuale per GeoUML è originale. Essa è definita in modo da fornire una definizione che corrisponde in maniera biunivoca alla forma grafica, cioè, dato uno schema in forma grafica è possibile derivarne la forma testuale e viceversa.

Per la specifica della sintassi della rappresentazione testuale è utilizzata la classica notazione delle grammatiche BNF con le seguenti convenzioni:

- l'assioma della grammatica è <S> (l'assioma è il simbolo non terminale di partenza per la generazione delle frasi);
- i simboli terminali sono in corsivo sottolineato (per simboli terminali si intendono i simboli che fanno parte della frase finale);
- il simbolo | viene usato per indicare un'alternativa tra due simboli o sequenze di simboli;
- i simboli non terminali sono indicati tra parentesi acute <...> (i simboli non terminali sono simboli provvisori da sostituire applicando le regole della grammatica con altri simboli, eventualmente terminali).

Di seguito si riportano le regole principali della grammatica. Tali regole indicano che uno schema concettuale contiene una lista di elementi che possono rappresentare la definizione di: classi, domini, associazioni con attributi, vincoli topologici o vincoli di consistenza delle aggregazioni.

Regole generali della grammatica G

<S> → <classe>
<S> → <S> <elemento>
<elemento> → <classe> | <dominio> | <associazione con attributi> | <vincolo topologico> |
<vincolo di struttura> | <strato topologico> | <classe sottoarea> | <classe tratto> |
<classe evento lineare> | <classe evento puntiforme>

Le regole che definiscono la sintassi di ogni specifico <elemento> vengono riportate nei paragrafi che presentano il corrispondente costrutto.

N.B.: Alcune regole vengono presentate in modo incrementale, mostrando prima una o più versioni semplificate e successivamente la versione completa. Le versioni semplificate sono precedute da un asterisco. L'intera grammatica è riportata in Appendice 2.

2.1 Classe

La nozione di ***oggetto (o entità)*** fa riferimento a un elemento della realtà al quale è utile e sensato agganciare informazioni in forma di attributi e/o di associazioni con altri oggetti. Una ***classe*** definisce le proprietà comuni a un insieme di oggetti omogenei.

Definizione 1. Classe

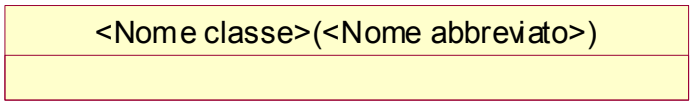
Una classe definisce le proprietà comuni di un insieme di oggetti. Tali proprietà possono essere attributi, operazioni o ruoli in associazioni con altre classi. In ogni istante di vita della base di dati geografica una classe ha un'estensione ext(C) costituita dall'insieme di oggetti, contenuti nella base di dati, che appartengono alla classe.

Rappresentazione testuale

* <classe> → *classe* <identificatore classe> <proprietà della classe> **.**
<identificatore classe> → <identificatore> *(* <identificatore abbreviato> *)* | <identificatore>
<identificatore abbreviato> → <identificatore>
<proprietà della classe> → *attributi:* <lista attributi> <ruoli> <vincoli>

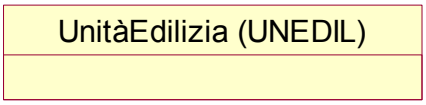
le regole che definiscono <identificatore> sono riportate in Appendice 2.

Rappresentazione grafica



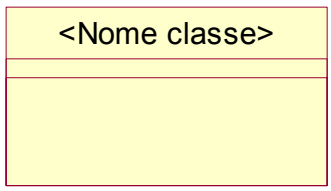
Esempio

classe UnitàEdilizia (UNEDIL)
<proprietà della classe>.



Traduzione in UML

Ogni classe <Nome classe> di GeoUML si traduce in una classe ("class") del linguaggio UML.



2.2 Attributo

La nozione di ***attributo*** fa riferimento ad una proprietà di una classe che risulta esprimibile attraverso un valore scelto in un dominio.

Definizione 2. Attributo

Un attributo a di dominio D associato ad una classe C è una funzione così definita:

$$a: ext(C) \rightarrow D.$$

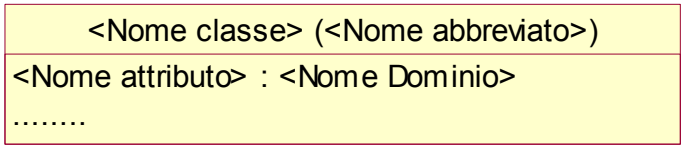
I domini di base per gli attributi non geometrici sono: String, Integer e Real. Poiché si tratta di una funzione il valore dell'attributo è definito su ogni istanza appartenente a ext(C).

Rappresentazione testuale (le regole fuori dal riquadro sono già state presentate in precedenza)

* <classe> → *classe* <identificatore classe> <proprietà della classe> **.**
<identificatore classe> → <identificatore> *(* <identificatore abbreviato> *)* | <identificatore>
<proprietà della classe> → *attributi:* <lista attributi> <ruoli> <vincoli>

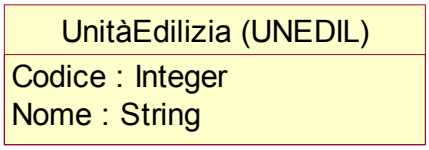
<lista attributi> → <attributo>
<lista attributi> → <attributo> **;** <lista attributi>
<attributo> → <attributo alfa> | <attributo geo>
* <attributo alfa> → <nome attributo> **:** <dominio alfa>
<nome attributo> → <identificatore>
* <dominio alfa> → <identificatore> | *String* | *Integer* | *Real*

Rappresentazione grafica



Esempio

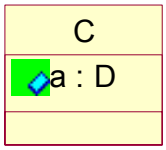
classe UnitàEdilizia (UNEDIL)
attributi: Codice: *Integer*; Nome: *String*.



Traduzione in UML

Ogni attributo a di dominio D (D∈{String, Real, Integer}) definito su una classe C di GeoUML si traduce in un attributo della classe ("class") del linguaggio UML che corrisponde alla classe C con le seguenti caratteristiche UML:

- Visibility: public (visibile all'esterno della classe: rappresentato dal simbolo azzurro),
- Multiplicity: 1..1 (ogni attributo ha uno e un sol valore: è l'impostazione di default).



Ogni dominio di base di GeoUML ha il suo corrispondente in UML. La corrispondenza è per nome.

2.3 Attributo enumerato

Un ***attributo enumerato*** è un attributo con dominio finito i cui valori sono predefiniti ed elencati nello schema.

Definizione 3. Attributo enumerato

Un attributo enumerato a di dominio $D=\{v_1, ..., v_n\}$ associato ad una classe C è una funzione così definita:

$$a: ext(C) \rightarrow D.$$

La rappresentazione testuale può assumere due forme: nella prima il dominio dell'attributo viene definito direttamente nella specifica dell'attributo stesso, elencando l'insieme dei suoi valori; nella secondo forma si definisce un dominio esplicito di cui si usa il nome nella definizione dell'attributo. Nella rappresentazione grafica la prima forma viene rappresentata come la seconda introducendo un dominio dal nome Dom_<nome classe>_<nome attributo>.

Rappresentazione testuale (le regole fuori dal riquadro sono già state presentate in precedenza)

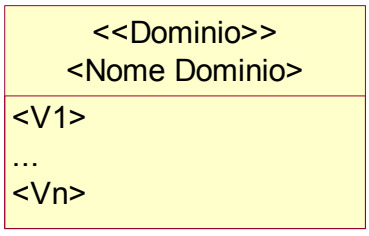
* <attributo alfa> → <nome attributo> *:* <dominio attributo>
<nome attributo> → <identificatore>

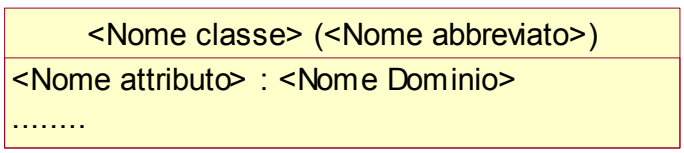
<dominio alfa> → <identificatore> | <dominio enumerato> | *String* | *Integer* | *Real*
<dominio enumerato> → *(* <lista valori> *)*
<lista valori> → <valore>
<lista valori> → <valore> **,** <lista valori>
<valore> → <identificatore>

Se si è usata la seconda forma, deve essere definito separatamente il dominio applicando la seguente regola:

<dominio> → *dominio* <identificatore dominio> **:** <dominio enumerato> **.**

Rappresentazione grafica (seconda forma)

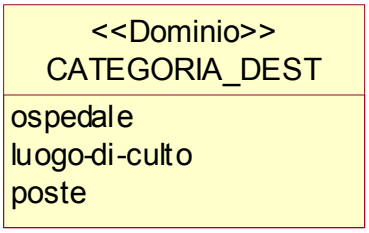


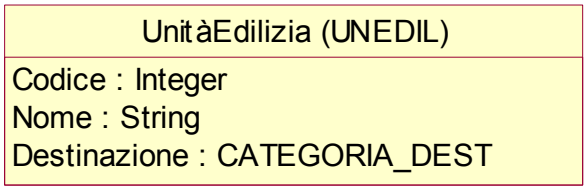


Esempio

classe UnitàEdilizia (UNEDIL)
attributi:
Codice: *Integer*;
Nome: *String*;
Destinazione: CATEGORIA_DEST.

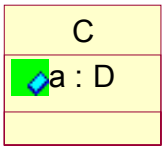
dominio CATEGORIA_DEST: *(*ospedale, luogo-di-culto, poste*)*.

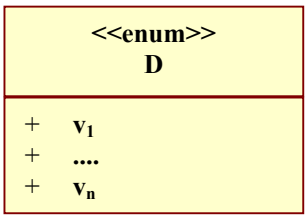




Traduzione in UML

Ogni attributo enumerato a di dominio D = (v_1, ..., v_n) definito su una classe C di GeoUML si traduce in un attributo della classe ("class") del linguaggio UML che corrisponde alla classe C con l'aggiunta di una classe streotipata "<<enum>>" per la rappresentazione del dominio D. Le caratteristiche dell'attributo a coincidono con quelle già precisate per l'attributo normale. Nel caso in cui il dominio non abbia un nome nello Schema Concettuale, in UML si impone il nome C_a (vale a dire, <Nome classe>_<Nome attributo>)





2.4 Cardinalità degli attributi (attributo multivalore)

Un attributo prevede che ogni oggetto istanza di una classe abbia un valore e che tale valore sia unico.
Tale caratteristica dell'attributo può essere modificata specificando esplicitamente la cardinalità dell'attributo.

Definizione 4. Cardinalità di un attributo

La cardinalità di un attributo definisce il numero minimo e il numero massimo di valori che l'attributo può assumere. Si indica sintatticamente nel seguente modo: [min..max], dove min e max indicano la cardinalità minima e la cardinalità massima dell'attributo. Il valore di default (vale a dire, la cardinalità attribuita in caso di assenza della esplicita indicazione [min..max]) è [1..1].

I casi che si possono verificare sono in particolare i seguenti:

cardinalità	significato
[1..1]	Attributo obbligatorio ad un sol valore
[0..1]	Attributo opzionale ad un sol valore
[1..*]	Attributo obbligatorio e multivalore
[0..*]	Attributo opzionale e multivalore
[1..<num>]	Attributo obbligatorio con al massimo <num> valori (<num> maggiore di 1)
[0..<num>]	Attributo opzionale con al massimo <num> valori (<num> maggiore di 1)
[<n_min>..<n_max>]	Attributo obbligatorio con al minimo <n_min> valori e al massimo <n_max> valori (<n_min> maggiore di 1, e <n_max> maggiore di <n_min>)

Rappresentazione testuale

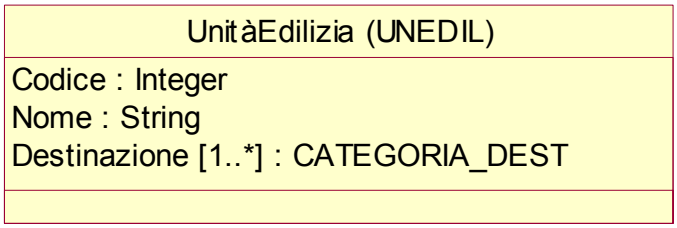
* <attributo alfa> → <nome attributo> <cardinalità> **:** <dominio alfa>
<cardinalità> → ε
<cardinalità> → *[* <card min>**..**<card max> *]*
<card min> → *0* | <cifra diversa da zero>
<card min> → <numero> <cifra>
<numero> → <numero> <cifra>
<numero> → <cifra diversa da zero>
<card max> → * | <cifra diversa da zero>
<card max> → <numero> <cifra>

Rappresentazione grafica

<Nome classe> (<Nome abbreviato>)
<Nome attributo>[<min> .. <max>] : <Nome Dominio>

Esempio

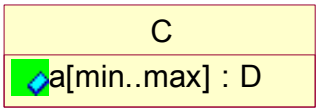
classe UnitàEdilizia (UNEDIL)
attributi:
Codice: *Integer*;
Nome: *String*;
Destinazione [1..*]: CATEGORIA_DEST.



Traduzione in UML

Ogni attributo a di dominio D definito su una classe C di GeoUML con cardinalità [min..max] si traduce in un attributo della classe ("class") del linguaggio UML che corrisponde alla classe C. Le caratteristiche dell'attributo a sono:

- Visibility: public (visibile all'esterno della classe: rappresentato dal simbolo azzurro),
- Multiplicity: [min..max].



2.5 Attributo geometrico

L'***attributo geometrico*** è un attributo che ha come dominio un dominio geometrico. I domini geometrici ammessi dal modello sono costituiti dalle classi di seguito definite e ottenute come specializzazioni delle classi dello standard ISO 19107 (Spatial Schema).

Definizione 5. Attributo geometrico

Un attributo geometrico g di dominio geometrico D_G associato ad una classe C è una funzione così definita:

$$g: ext(C) \rightarrow D_G.$$

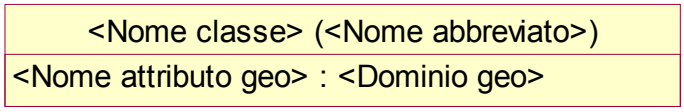
I domini possibili per gli attributi geometrici sono costituiti dalle classi di seguito definite (si veda la sottosezione 2.5.1) e ottenute come specializzazione delle classi contenute nella gerarchia con radice in GM_Object dello standard ISO 19107 "Spatial schema" (vedi Figura 2).

Rappresentazione testuale (le regole fuori dal riquadro sono già state presentate in precedenza)

<attributo> → <attributo alfa> | <attributo geo>

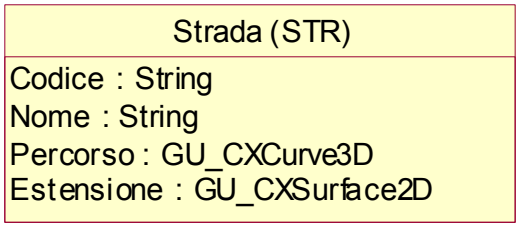
* <attributo geo> → <attributo geo> → <nome attributo> <cardinalità> **:** <dominio geo>
<dominio geo> → *GU_Point2D* | *GU_Point3D* | *GU_CPCurve2D* | *GU_CPCurve3D* | *GU_Ring2D* | *GU_Ring3D* | *GU_CPSurface2D* | *GU_CXCurve2D* | *GU_CXCurve3D* | *GU_CXRing2D* | *GU_CXRing3D* | *GU_CXSurface2D* | *GU_Complex2D* | *GU_Complex2D* | *GU_Aggregate2D* | *GU_Aggregate3D* | *GU_MPoint2D* | *GU_MPoint3D* | *GU_MCurve2D* | *GU_MCurve3D* | *GU_MSurface2D* | *GU_MRing2D* | *GU_MRing3D* | *GU_CNCurve2D* | *GU_CPSurfaceB3D* | *GU_CXSurfaceB3D* | *GU_CNCurve3D*

Rappresentazione grafica



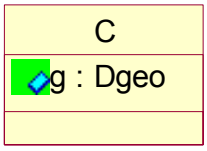
Esempio

classe Strada (STR):
attributi:
Codice: *String*;
Nome: *String*;
Percorso: *GU_CXCurve3D*;
Estensione: *GU_CXSurface2D*.



Traduzione in UML

Ogni attributo geometrico g di dominio D_{geo} (D_{geo} è una classe ammessa come dominio geometrico dal modello; tali classi sono caratterizzate dal prefisso "GU_") definito su una classe C di GeoUML si traduce in un attributo della classe ("class") del linguaggio UML che corrisponde alla classe C. Il tipo dell'attributo corrisponde alla classe del modello indicata nella classe C come dominio di g. Le caratteristiche dell'attributo g coincidono con quelle già precisate per l'attributo normale.



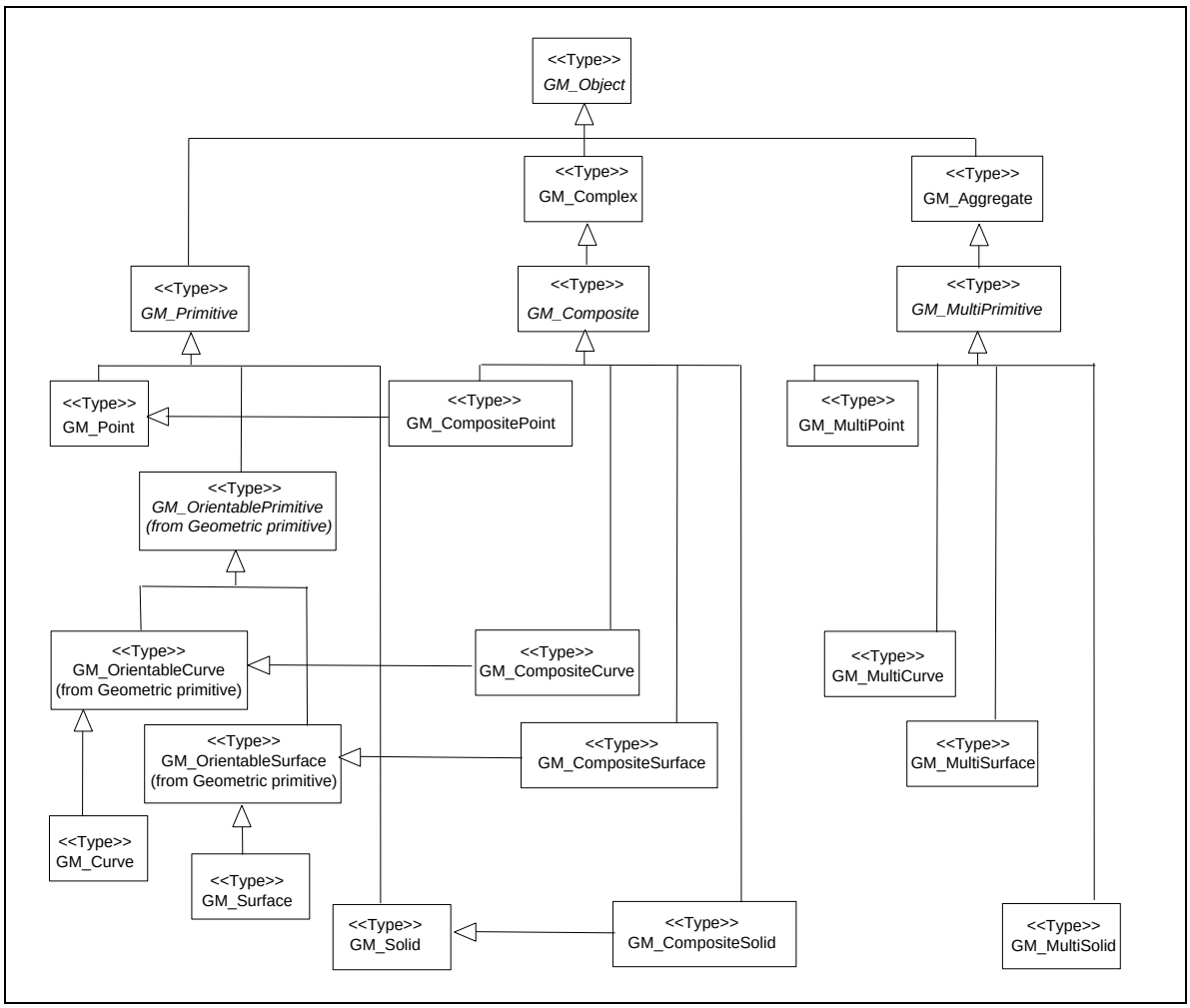


Figura 2 : Gerachia delle sottoclassi di GM_Object come definita nello standard ISO 19107.

2.5.1 Specializzazioni dello Spatial schema

Rispetto allo Spatial Schema di ISO/TC 211 GeoUML contiene sia restrizioni pure (eliminazione di classi) sia delle specializzazioni di classi.

Non sono state adottare direttamente le classi dello Spatial Schema per i seguenti motivi:

- è necessario distinguere le classi contenenti oggetti del piano (2D) e le classi contenenti oggetti dello spazio (3D);
- è importante selezionare dallo Spatial Schema solo l'insieme delle classi che possono essere usate da utenti inesperti nella definizione di schemi di basi di dati geografiche, in modo da evitare scelte errate;
- è necessario definire alcune specializzazioni della classe GM_Complex per rappresentare complessi costituiti da soli punti, sole linee o da sole superfici;
- è necessario introdurre due classi aggiuntive per la rappresentazione di superfici per le quali si vuole memorizzare l'estensione nel piano e la frontiera (o boundary) nello spazio 3D, mantenendo la coerenza delle due rappresentazioni (superfici B3D).

Nei tre paragrafi che seguono si descrivono tutte le classi introdotte in GeoUML per la rappresentazione dei domini geometrici. Nel primo paragrafo si descrivono le classi per le quali si è applicata solo la specializzazione rispetto allo spazio di riferimento (2D o 3D), nel secondo si descrivono le due specializzazioni della classe GM_Complex e nel terzo si descrivono le due classi aggiuntive per le superfici B3D. In Figura 3 si mostra l'elenco di tutte le classi geometriche di GeoUML.

2.5.1.1 Specializzazioni basate sullo spazio di riferimento

In questo paragrafo si presentano le classi geometriche di GeoUML, vale a dire, quelle classi che devono essere usate come domini degli attributi geometrici presenti in uno schema GeoUML. In particolare si riportano qui le classi che non richiedono definizioni particolarmente complesse, mentre nei due successivi paragrafi si presentano quelle classi che richiedono una trattazione separata.

Ogni classe è figlia di una classe dello Spatial Schema (ISO 19107) ed eredita quindi dalla classe tutte le proprietà definite nello standard aggiungendo eventualmente vincoli e/o proprietà. In questo paragrafo si riportano le classi che non richiedono definizioni particolarmente complesse,

Alcune proprietà delle classi geometriche di GeoUML sono definite sulla classe radice della gerarchia di classi proposta nello Spatial Schema. In tale classe, detta GM_Object, si riportano le proprietà comuni a tutte le classi. In particolare tra le proprietà comuni a tutte le classi troviamo la frontiera (o boundary). Di seguito si riportano alcune di queste proprietà.

GM_Object

GM_Object::isCycle(): Boolean
GM_Object::isSimple(): Boolean
GM_Object::boundary( ): GM_boundary
GM_Boundary: { isCycle() =TRUE }
GM_Object::CRS():CS_CRS

Si elencano di seguito le classi geometriche di GeoUML indicando la classe padre dello spatial schema, alcune delle proprietà ereditate e le proprietà o vincoli aggiuntivi specifici della classe di GeoUML. Per quasi tutte le classi esistono due versioni: quella nello spazio a due dimensioni (2D) e quella nello spazio a tre dimensioni (3D).

Gli spazi di riferimento si suppongono definiti in ogni schema scritto in GeoUML attraverso la specifica di due oggetti della classe CS_CRS (Coordinate Reference System) che chiameremo GU_CRS-2D e GU_CRS-3D, per i quali, secondo il documento dello standard ISO 19111, vanno specificati i seguenti attributi:

- *validArea*: porzione di territorio dove il sistema si applica;
- *datum.type*: geodetic, vertical or engineering
- *datum.datumID*: identificatore del datum,
- *datum.alias*: nome del datum
- *datum.point*: coordinate del punto o dei punti ancora per il datum
- *datum.realizationEpoch*: data di realizzazione del datum
- *primeMeridian.meridianID*: identificatore del meridiano primo
- *ellipsoid.ellipsoidID*: identificatore dell'ellissoide
- *coordinateSystem.CSID*: identificatore del sistema di riferimento
- *coordinateSystem.type*: Cartesian, geodetic, projected, polar, ecc..
- *coordinateSystem.dimension*: numero di coordinate

Inoltre si suppone presente in tutte le classi un'operazione aggiuntiva detta *planar()* che definiamo quindi sulla classe GU_Object specializzazione della classe GM_Object:

GU_Object::planar( ): GU_Object
{ self.planar() = self.transform(GU_CRS-2D) }

Tale funzione permette di calcolare, dato un valore di una classe che contiene oggetti geometrici dello spazio 3D, la proiezione di tale valore nello spazio 2D. Si tratta quindi di una ridefinizione dell'operazione transform già presente sulla classe GM_Object.

Definizione 6. Punti in 2D (*GU_Point2D*)

Eredita dalla classe dello Spatial Schema (ISO 19107): GM_Point

Vincoli:

context *GU_Point2D*

inv: *{self.CRS = GU_CRS-2D }*
{self.mbRegion() = NULL}
{self.representativePoint() = self.Position()}
{self.boundary().isEmpty = TRUE}
{self.dimension() = 0}
{self.coordinateDimension() = 2}
{self.envelope() = NULL}
{self.centroid() = self.Position()}
{self.convexHull() = NULL}

Proprietà ereditate ridefinite:

- *GU_Point2D::**closure**(): GU_CXPoint2D*
- *GU_Point2D::**buffer**(radius: Distance): GU_CPSurface2D*
- *GU_Point2D::**planar**(): GU_Point2D*
  {self.planar() = self}

Definizione 7. Punti in 3D (*GU_Point3D*)

Eredita dalla classe dello Spatial Schema (ISO 19107): GM_Point

Vincoli:

context *GU_Point3D*

inv: *self.CRS = GU_CRS-3D }*
{self.mbRegion() = NULL}
{self.representativePoint() = self.Position()}
{self.boundary().isEmpty = TRUE}
{self.dimension() = 0}
{self.coordinateDimension() = 3}
{self.envelope() = NULL}
{self.centroid() = self.Position()}
{self.convexHull() = NULL}

Proprietà ereditate ridefinite:

- *GU_Point3D::**closure**(): GU_CXPoint3D*
- *GU_Point3D::**buffer**(radius: Distance): GM_Solid*
- *GU_Point3D::**planar**(): GU_Point2D*

Definizione 8. Composite Curve in 2D (*GU_CPCurve2D*)

Eredita dalla classe dello Spatial Schema (ISO 19107): GM_CompositeCurve

Vincoli:

context *GU_CPCurve2D*

inv: *{self.CRS = GU_CRS-2D}*
{self.dimension() = 1}
{self.coordinateDimension() = 2}
{self.element → forall(a: GM_Primitive | a.OclIsTypeOf(GM_Curve) implies
a.isCycle() = FALSE)}

Si noti che non viene imposto il vincolo: {self.isSimple() = TRUE}, tuttavia si impone invece che le curve che compongono una "composite curve" siano aperte (*a.isCycle() = FALSE*) . Ciò evita che, in caso di aggiunta di una nuova curva primitiva connessa ad una primitiva chiusa di una "composite curve", sia necessario spezzare le primitive chiuse, per renderle aperte, prima di inserire la nuova curva primitiva.

Proprietà ereditate ridefinite:

- *GU_CPCurve2D:: **mbRegion**(): GU_CPSurface2D*
- *GU_CPCurve2D:: **envelope**(): GU_CPSurface2D*
- *GU_CPCurve2D::**buffer**(radius: Distance): GU_CXSurface2D*
- *GU_CPCurve2D:: **boundary**(): GU_CurveBoundary2D*
- *GU_CurveBoundary2D:: **startpoint**(): reference <GU_Point2D>*
- *GU_CurveBoundary2D:: **endpoint**(): reference <GU_Point2D>*

- *GU_CPCurve2D::**planar**(): GU_CPCurve2D*
  {self.planar() = self}

Definizione 9. Composite Curve in 3D (*GU_CPCurve3D*)

Eredita dalla classe dello Spatial Schema (ISO 19107): GM_CompositeCurve

Vincoli:
***context** GU_CPCurve3D*
***inv**: {self.CRS = GU_CRS-3D }*
{self.dimension() = 1}
{self.coordinateDimension() = 3}
{self.element →forall(a: GM_Primitive | a.OclIsTypeOf(GM_Curve) implies a.isCycle() = FALSE)}

Vale l'osservazione fatta per le GU_CPCurve2D.

Proprietà ereditate ridefinite:

- *GU_CPCurve3D:: **mbRegion**(): GM_Solid*
- *GU_CPCurve3D:: **envelope**(): GM_Solid*
- *GU_CPCurve3D::**buffer**(radius: Distance): GM_Complex*
- *GU_CPCurve3D::**planar**(): GU_ CXCurve2D*
- *GU_CPCurve3D:: **boundary**( ): GU_CurveBoundary3D*
  GU_CurveBoundary3D è una classe ausiliaria che eredita da GM_CurveBoundary, dove sono ridefinite le seguenti proprietà:
  *GU_CurveBoundary3D:: **startpoint**: reference <GU_Point3D>*
  *GU_CurveBoundary3D:: **endpoint**: reference <GU_Point3D>*

Definizione 10. Ring in 2D (*GU_CPRing2D*)

Eredita dalla classe: GU_CPCurve2D

Vincoli:
***context** GU_CPRing2D*
***inv** : {self.isCycle =TRUE}*
{self.isSimple =TRUE}

Definizione 11. Ring in 3D (*GU_CPRing3D*)

Eredita dalla classe: GU_CPCurve3D

Vincoli:
***context** GU_CPRing3*

inv *: {self.isCycle =TRUE}*
{self.isSimple =TRUE}

Definizione 12. Composite Surface in 2D (*GU_CPSurface2D*)

Eredita dalla classe dello Spatial Schema (ISO 19107): GM_CompositeSurface

Vincoli:
context *GU_CPSurface2D*
inv: *{self.CRS = GU_CRS-2D}*
{self.dimension() = 2}
{self.coordinateDimension() = 2}

Proprietà ereditate ridefinite:

- *GU_CPSurface2D:: **mbRegion**(): GU_CPSurface2D*
- *GU_CPSurface2D:: **envelope**(): GU_CPSurface2D*
- *GU_CPSurface2D::**buffer**(radius: Distance): GU_CXSurface2D*
- *GU_CPSurface2D::**planar**(): GU_CPSurface2D*
  {self.planar() = self}
- *GU_CPSurface2D:: **boundary**: GU_SurfaceBoundary2D*
  GU_CurveBoundary2D è una classe ausiliaria che eredita da GM_SurfaceBoundary, dove sono ridefinite le seguenti proprietà:
  *GU_SurfaceBoundary2D:: **exterior**[0, 1] : GU_CPRing2D*
  *GU_SurfaceBoundary2D:: **interior**[0..*] : GU_CPRing2D*

Definizione 13. Aggregate in 2D (*GU_Aggregate2D*)

Eredita dalla classe dello Spatial Schema (ISO 19107): GM_Aggregate
Un oggetto della classe è costituito da un insieme di oggetti delle classi GU_Point2D, GU_CPCurve2D e GU_CPSurface2D.

Vincoli:
context *GU_Aggregate2D*
inv: *{self.element→forall(a: GM_Object | a.OclIsTypeOf(GU_Point2D) or a.OclIsTypeOf(GU_CPCurve2D) or a.OclIsTypeOf(GU_CPSurface2D)}*

Proprietà ereditate ridefinite:

- *GU_Aggregate2D:: **mbRegion**(): GU_CPSurface2D*
- *GU_Aggregate2D:: **envelope**(): GU_CPSurface2D*
- *GU_Aggregate2D::**buffer**(radius: Distance): GU_CXSurface2D*
- *GU_Aggregate2D::**planar**(): GU_Aggregate2D*
  {self.planar() = self}

Definizione 14. Aggregate in 3D (*GU_Aggregate3D*)

Eredita dalla classe dello Spatial Schema (ISO 19107): GM_Aggregate

Un oggetto della classe è costituito da un insieme di oggetti delle classi GU_Point3D e GU_CPCurve3D.

Vincoli:

***context** GU_Aggregate3D*

***inv**: {self.element→forall(a: GM_Object | a.OclIsTypeOf(GU_Point3D) or a.OclIsTypeOf(GU_CPCurve3D)}*

Proprietà ereditate ridefinite:

- *GU_Aggregate3D:: **mbRegion**(): GM_Solid*
- *GU_Aggregate3D:: **envelope**(): GM_Solid*
- *GU_Aggregate3D::**buffer**(radius: Distance): GM_Complex*
- *GU_Aggregate3D::**planar**(): GU_Aggregate2D*

Definizione 15. MultiPoint in 2D (*GU_MPoint2D*)

Eredita dalla classe dello Spatial Schema (ISO 19107): GM_MultiPoint

Un oggetto della classe è costituito da un insieme di oggetti della classe GU_Point2D.

Vincoli:

***context** GU_MPoint2D*

***inv**: {self.element→forall(a: GM_Object | a.OclIsTypeOf(GU_Point2D)}*

Proprietà ereditate ridefinite:

- *GU_MPoint2D:: **mbRegion**(): GU_CPSurface2D*
- *GU_MPoint2D:: **envelope**(): GU_CPSurface2D*
- *GU_MPoint2D::**buffer**(radius: Distance): GU_CXSurface2D*
- *GU_MPoint2D::**planar**(): GU_MPoint2D*
  {self.planar() = self}

Definizione 16. MultiPoint in 3D (*GU_MPoint3D*)

Eredita dalla classe dello Spatial Schema (ISO 19107): GM_MultiPoint

Un oggetto della classe è costituito da un insieme di oggetti della classe GU_Point3D.

Vincoli:

***context** GU_MPoint3D*

***inv**: {self.element→forall(a: GM_Object | a.OclIsTypeOf(GU_Point3D)}*

Proprietà ereditate ridefinite:

- *GU_MPoint3D:: **mbRegion**(): GM_Solid*
- *GU_MPoint3D:: **envelope**(): GM_Solid*
- *GU_MPoint3D::**buffer**(radius: Distance): GM_Complex*
- *GU_MPoint3D::**planar**(): GU_MPoint2D*

Definizione 17. MultiCurve in 2D (*GU_MCurve2D*)

*Eredita dalla classe dello Spatial Schema (ISO 19107):*GM_MultiCurve
Un oggetto della classe è costituito da un insieme di oggetti della classe GU_CPCurve2D.

Vincoli:
***context** GU_MCurve2D*
***inv**: {self.element→forall(a: GM_Object | a.OclIsTypeOf(GU_CPCurve2D)}*

Proprietà ereditate ridefinite:

- *GU_MCurve2D:: **mbRegion**(): GU_CPSurface2D*
- *GU_MCurve2D:: **envelope**(): GU_CPSurface2D*
- *GU_MCurve2D::**buffer**(radius: Distance): GU_CXSurface2D*
- *GU_MCurve2D::**planar**(): GU_MCurve2D*
  {self.planar() = self}

Definizione 18. MultiCurve in 3D (*GU_MCurve3D*)

Eredita dalla classe dello Spatial Schema (ISO 19107): GM_MultiCurve
Un oggetto della classe è costituito da un insieme di oggetti della classe GU_CPCurve3D.

Vincoli:
***context** GU_MCurve3D*
***inv**: {self.element→forall(a: GM_Object | a.OclIsTypeOf(GU_CPCurve3D)}*

Proprietà ereditate ridefinite:

- *GU_MCurve3D:: **mbRegion**(): GM_Solid*
- *GU_MCurve3D:: **envelope**(): GM_Solid*
- *GU_MCurve3D::**buffer**(radius: Distance): GM_Complex*
- *GU_MCurve3D::**planar**(): GU_MCurve2D*

Definizione 19. MultiSurface in 2D (*GU_MSurface2D*)

*Eredita dalla classe dello Spatial Schema (ISO 19107):*GM_MultiSurface
Un oggetto della classe è costituito da un insieme di oggetti della classe GU_CPCurve3D.

Vincoli:
***context** GU_MSurface2D*

***inv**: {self.element→forall(a: GM_Object | a.OclIsTypeOf(GU_CPSurface2D)}*

Proprietà ereditate ridefinite:

- *GU_MSurface2D:: **mbRegion**(): GU_CPSurface2D*
- *GU_MSurface2D:: **envelope**(): GU_CPSurface2D*
- *GU_MSurface2D::**buffer**(radius: Distance): GU_CXSurface2D*
- *GU_MSurface2D::**planar**(): GU_MSurface2D*
  {self.planar() = self}

Definizione 20. MultiRing in 2D (*GU_MRing2D*)

Eredita dalla classe: GU_MCurve2D

Vincoli:

***context** GU_MRing2D*

***inv**: {self.element→forall(a: GM_Object | a.OclIsTypeOf(GM_Curve) and a.isCycle()=TRUE and a.isSimple()=TRUE}*

Definizione 21. MultiRing in 3D (*GU_MRing3D*)

Eredita dalla classe: GU_MCurve3D

Vincoli:

***context** GU_MRing3D*

***inv**: {self.element→forall(a: GM_Object | a.OclIsTypeOf(GM_Curve)) and a.isCycle()=TRUE and a.isSimple()=TRUE }*

2.5.1.2 Specializzazioni della classe GM_Complex

In GeoUML sono definite le seguenti due specializzazioni del tipo GM_Complex:

- *GU_CXPoint2D* (*GU_CXPoint3D*),
- *GU_CXCurve2D* (*GU_CXCurve3D*),
- *GU_CXSurface2D*.

La prima rappresenta i complessi costituiti solo da punti (oggetti di tipo GM_Point), la seconda rappresenta i complessi costituiti solo da curve (oggetti di tipo GM_Curve) e dagli oggetti che costituiscono il loro boundary; la terza rappresenta i complessi costituiti solo da superfici (oggetti di tipo GM_Surface) e dagli oggetti che costituiscono il loro boundary.

Queste tre (in totale cinque considerando le distinzioni sulla dimensione dello spazio) nuove classi geometriche, specializzazioni di GM_Complex, rappresentano i complessi di dimensione omogenea (mentre i GM_Complex generici sono multidimensionali e quindi non dimensionalmente omogenei).

La necessità di queste specializzazioni si spiega nel modo seguente:

1. nello Spatial Schema ISO/TC 211 esistono solo i complessi multidimensionali oppure le primitive composte (rappresentate dalle classi GM_CompositeCurve e GM_CompositeSurface); le primitive composte sono complessi omogenei isomorfi alle primitive rappresentate dalle classi GM_Curve e GM_Surface e sembra che, nell'intenzione degli estensori dello Spatial Schema, la maggior parte degli oggetti composti dovrebbe appartenere a queste classi. In pratica (cfr. documento 1n1004) si è riscontrato che molti oggetti del mondo reale sono rappresentati da complessi omogenei che però NON sono isomorfi con singole primitive (ad esempio, una strada, intesa come l'insieme degli elementi associati allo stesso toponimo, nella forma lineare non è sempre isomorfa a una singola linea, perché può essere spezzettata o contenere biforcazioni; tuttavia essa è omogenea nella dimensione, cioè composta solamente da elementi lineari). Per questo motivo molti oggetti non possono essere definiti come "Composite...".
2. d'altra parte, definire in questi casi gli oggetti omogenei come generici GM_Complex (multidimensionali) comporta uno svantaggio fondamentale: come spiegato più avanti, per i complessi multidimensionali non è ben consolidata la definizione della frontiera (boundary), mentre per i complessi omogenei tale definizione può essere data; la definizione della frontiera è a sua volta fondamentale per definire i diversi tipi di relazioni spaziali tra oggetti. Per questi motivi è consigliabile non utilizzare, se possibile, il generico GU_Complex2D (o GU_Complex3D) (specializzazioni di GM_Complex) come dominio di un attributo geometrico.

La definizione in UML delle due nuove classi è riportata di seguito:

Definizione 22. Complex in 2D (*GU_Complex2D*)

Eredita dalla classe dello Spatial Schema (ISO 19107): GM_Complex

Vincoli:

context *GU_Complex2D*
inv: *{self.CRS = GU_CRS-2D}*
{self.coordinateDimension() = 2}

Proprietà ereditate ridefinite:

- *GU_Complex2D::* ***mbRegion****(): GU_CPSurface2D*
- *GU_Complex2D::* ***envelope****(): GU_CPSurface2D*
- *GU_Complex2D::****buffer****(radius: Distance): GU_CXSurface2D*
- *GU_Complex2D::****planar****(): GU_Complex2D*
  {self.planar() = self}

Definizione 23. Complex in 3D (*GU_Complex3D*)

Eredita dalla classe dello Spatial Schema (ISO 19107): GM_Complex
Un oggetto di questa classe è un complesso che non contiene superfici.

Vincoli:

context *GU_Complex3D*
inv: *{self.CRS = GU_CRS-3D}*
{self.element→forall(a: GM_Primitive | not a.OclIsTypeOf(GM_Surface))}

{self.coordinateDimension() = 3}

Proprietà ereditate ridefinite:

- *GU_Complex3D:: **mbRegion**(): GM_Solid*
- *GU_Complex3D:: **envelope**(): GM_Solid*
- *GU_Complex3D::**buffer**(radius: Distance): GM_Complex*
- *GU_Complex3D::**planar**(): GU_Complex2D*

Definizione 24. Complex Point in 2D (*GU_CXPoint2D*)

Eredita dalla classe: GU_Complex2D

Vincoli:

***context** GU_CXPoint2D*

***inv**: {self.element→forall(a: GM_Primitive | a.OclIsTypeOf(GU_Point2D))}*
{self.dimension() = 0}

Definizione 25. Complex Point in 3D (*GU_CXPoint3D*)

Eredita dalla classe: GU_Complex3D

Vincoli:

***context** GU_CXPoint3D*

***inv**: {self.element→forall(a: GM_Primitive | a.OclIsTypeOf(GU_Point3D))}*
{self.dimension() = 0}

Definizione 26. Complex Curve in 2D (*GU_CXCurve2D*)

Eredita dalla classe: GU_Complex2D

Vincoli:

***context** GU_CXCurve2D*

***inv**: {self.element →forall(a: GM_Primitive | a.dimension() ≤ 1) and*
self.element →forall(a: GM_Primitive | a.dimension() = 0 implies
self.element→ exist(b: GM_Primitive | b.boundary()→includes(a)))}
{self.dimension() = 1}

Proprietà ereditate ridefinite:

- *GU_CXCurve2D:: **boundary()**: GU_CXPoint2D*
  GU_CXPoint2D è una classe definita in precedenza ed eredita anche da GM_ComplexBoundary.

Definizione 27. Complex Curve in 3D (*GU_CXCurve3D*)

Eredita dalla classe: GU_Complex3D

Vincoli:

context *GU_CXCurve3D*

inv: *{self.element →forall(a: GM_Primitive | a.dimension() ≤ 1) and*
self.element →forall(a: GM_Primitive | a.dimension() = 0 implies
self.element→ exist(b: GM_Primitive | b.boundary()→includes(a)))}
{self.dimension() = 1}

Proprietà ereditate ridefinite:

- *GU_CXCurve3D:: **boundary()**: GU_CXPoint3D*
  GU_CXPoint3D è una classe definita in precedenza ed eredita anche da GM_ComplexBoundary.

Definizione 28. Complex Connected Curve in 2D (*GU_CNCurve2D*)

Eredita dalla classe: GU_CXCurve2D

Vincoli:

context *GU_CNCurve2D*

inv: *{self.element →forall(a: GM_Primitive |*
self.element →forall(b: GM_Primitive | a <> b implies
self.subComplex→ exist(c: GM_Complex |
c.OclIsTypeOf(GU_CPCurve2D) and
c.element→ includes(a) and c.element→ includes(b)))))}

Definizione 29. Complex Connected Curve in 3D (*GU_CNCurve3D*)

Eredita dalla classe: GU_CXCurve3D

Vincoli:

context *GU_CNCurve3D*

inv: *{self.element →forall(a: GM_Primitive |*
self.element →forall(b: GM_Primitive | a <> b implies
self.subComplex→ exist(c: GM_Complex |
c.OclIsTypeOf(GU_CPCurve3D) and
c.element→ includes(a) and c.element→ includes(b)))))}

Definizione 30. Complex Ring in 2D (*GU_CXRing2D*)

Eredita dalla classe: GU_CXCurve2D

Vincoli:

context *GU_CXRing2D*

inv: *{self.subComplex→ forall(a: GM_Complex | a.OclIsTypeOf(GU_CPCurve2D) implies self.subComplex→ exists(b: GM_Complex | b.OclIsTypeOf(GU_Ring2D) and b.subComplex→ includes(a))) }*

Definizione 31. Complex Ring in 3D (*GU_CXRing3D*)

Eredita dalla classe: GU_CXCurve3D

Vincoli:

context *GU_CXRing3D*

inv: *{self.subComplex→ forall(a: GM_Complex | a.OclIsTypeOf(GU_CPCurve3D) implies self.subComplex→ exists(b: GM_Complex | b.OclIsTypeOf(GU_Ring3D) and b.subComplex→ includes(a))) }*

Definizione 32. Complex Surface in 2D (*GU_CXSurface2D*)

Eredita dalla classe dello Spatial Schema (ISO 19107): GM_Complex2D

Vincoli:

context *GU_CXSurface2D*

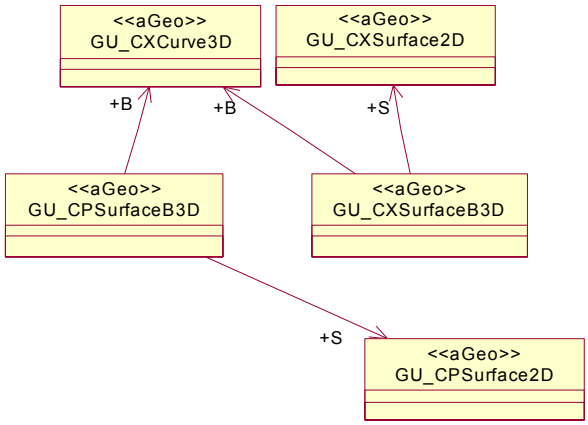
inv: *{self.element →forall(a: GM_Primitive | a.dimension()≤ 2) and*
self.element →forall(a: GM_Primitive | a.dimension()≤ 1 implies
self.element→ exist(b: GM_Primitive | b.dimension() = 2 and
b.boundary()→includes(a)))}

Proprietà ereditate ridefinite:

- *GU_CXSurface2D:: **boundary**: GU_CXRing2D*
  GU_CXRing2D è una classe definita precedentemente ed eredita anche da GM_ComplexBoundary.

2.5.1.3 Classi per la rappresentazione di superfici 2D con frontiera in 3D

Poichè nel DB topografico del progetto Intesa è prevista la rappresentazione di oggetti che vengono descritti come superfici dove la frontiera o boundary della superficie è nello spazio 3D mentre la superficie è nello spazio 2D, si sono introdotte delle classi specifiche per la rappresentazione di tali oggetti. Ovviamente tali classi consentono di descrivere separatamente le due rappresentazioni richiedendo che sia soddisfatto un vincoli di consistenza tra le due, vale a dire: la proiezione della curva 3D che rappresenta il boundary della superficie deve essere contenuta nel boundary della superficie 2D.
Le due classi differiscono solo per il tipo di rappresentazione della superficie 2D che in un caso è un composto e nell'altro è un complesso.
Di seguito si riporta la definizione delle due classi.



Come si nota le due classi partecipano a due associazioni monodirezionali ciascuna che le legano a: GU_CXCurve3D e a GU_CPSurface2D per GU_CPSurfaceB3D o a GU_CXSurface2D per GU_CXSurfaceB3D. Esiste inoltre un vincolo di consistenza tra le due rappresentazioni che si può definire in OCL nel seguente modo:

context *GU_CPSurfaceB3D (oppure GU_CXSurfaceB3D)*
inv: *{ check(self.B.planar(), {EQUAL}, self.S.boundary())}*

La funzione check usata in questa definizione viene specificata in OCL nel paragrafo 3.5

Classe di GeoUML	Classe padre di GeoUML	Classe padre di Spatial Schema
Classi che specializzano GM_Point		
GU_Point2D		GM_Point
GU_Point3D		GM_Point
Classi che specializzano GM_Composite		
GU_CPCurve2D		GM_CompositeCurve
GU_CPCurve3D		GM_CompositeCurve
GU_CPRing2D	GU_CPCurve2D	
GU_CPRing3D	GU_CPCurve3D	
GU_CPSurface2D		GM_CompositeSurface
Classi che specializzano GM_Complex		
GU_Complex2D		GM_Complex
GU_Complex3D		GM_Complex
GU_CXPoint2D	GU_Complex2D	
GU_CXPoint3D	GU_Complex3D	
GU_CXCurve2D	GU_Complex2D	
GU_CXCurve3D	GU_Complex3D	
GU_CNCurve2D	GU_CXCurve2D	
GU_CNCurve3D	GU_CXCurve3D	
GU_CXRing2D	GU_CXCurve2D	
GU_CXRing3D	GU_CXCurve3D	
GU_CXSurface2D	GU_Complex2D	
Classi che specializzano GM_Aggregate		
GU_Aggregate2D		GM_Aggregate
GU_Aggregate3D		GM_Aggregate
GU_MPoint2D		GM_MultiPoint
GU_MPoint3D		GM_MultiPoint
GU_MCurve2D		GM_MultiCurve
GU_MCurve3D		GM_MultiCurve
GU_MSurface2D		GM_MultiSurface
GU_MRing2D	GU_MCurve2D	
GU_MRing3D	GU_MCurve3D	
Classi che rappresentano superfici 2D/3D		
GU_CPSurfaceB3D		
GU_CXSurfaceB3D		

Figura 3 Tabella riassuntiva delle classi geometriche di GeoUML.

2.5.2 La frontiera (boundary) degli oggetti rappresentati nelle classi dello Spatial Schema

Le classi dello Spatial Schema per le quali lo standard ISO 19107 non specifica una definizione precisa di boundary sono: GM_Aggregate e GM_Complex (vedi Figura 4 e Figura 5).

Le proprietà del boundary degli oggetti di queste due classi sono definite nella classe astratta GM_Object e sono le seguenti:

- *GM_Object::boundary():GM_Boundary* (ISO 19107 - 6.2.2.4): tale segnatura dell'operazione boundary() indica che il risultato dell'operazione è sempre un oggetto della classe GM_Boundary
- *GM_Boundary è sottoclasse di GM_Complex* (ISO 19107 - 6.3.2): ciò significa che tutti gli oggetti prodotti dall'applicazione dell'operazione boundary() sono oggetti della classe GM_Complex.
- *boundary() → select(dimension) <= self.dimension - 1* (ISO 19107 - 6.2.2.4): ciò significa che tutti gli oggetti contenuti nel boundary di un oggetto O hanno dimensione di uno inferiore a quella dell'oggetto O.

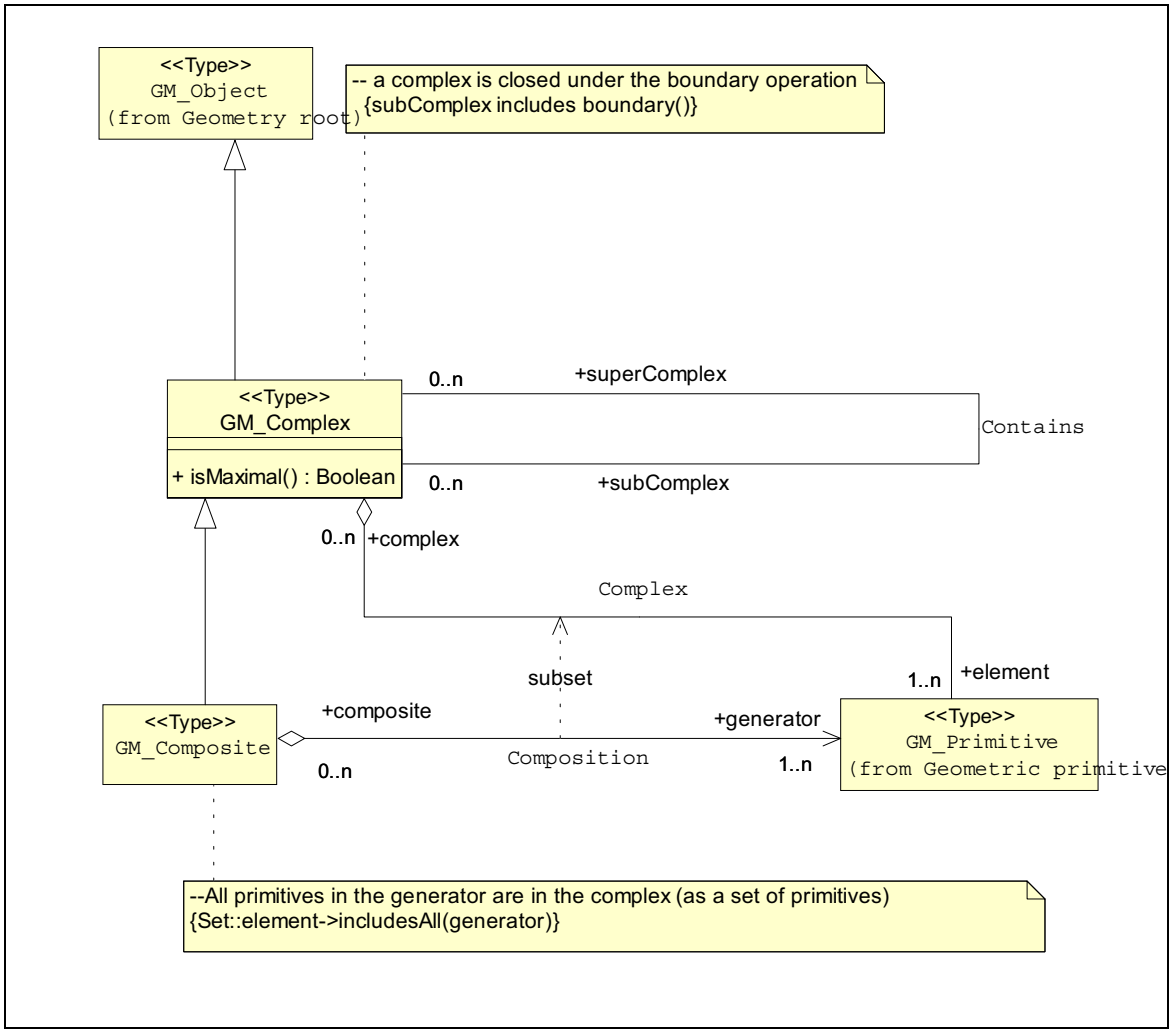


Figura 4 Il tipo GM_Complex come definito nello standard ISO 19107.

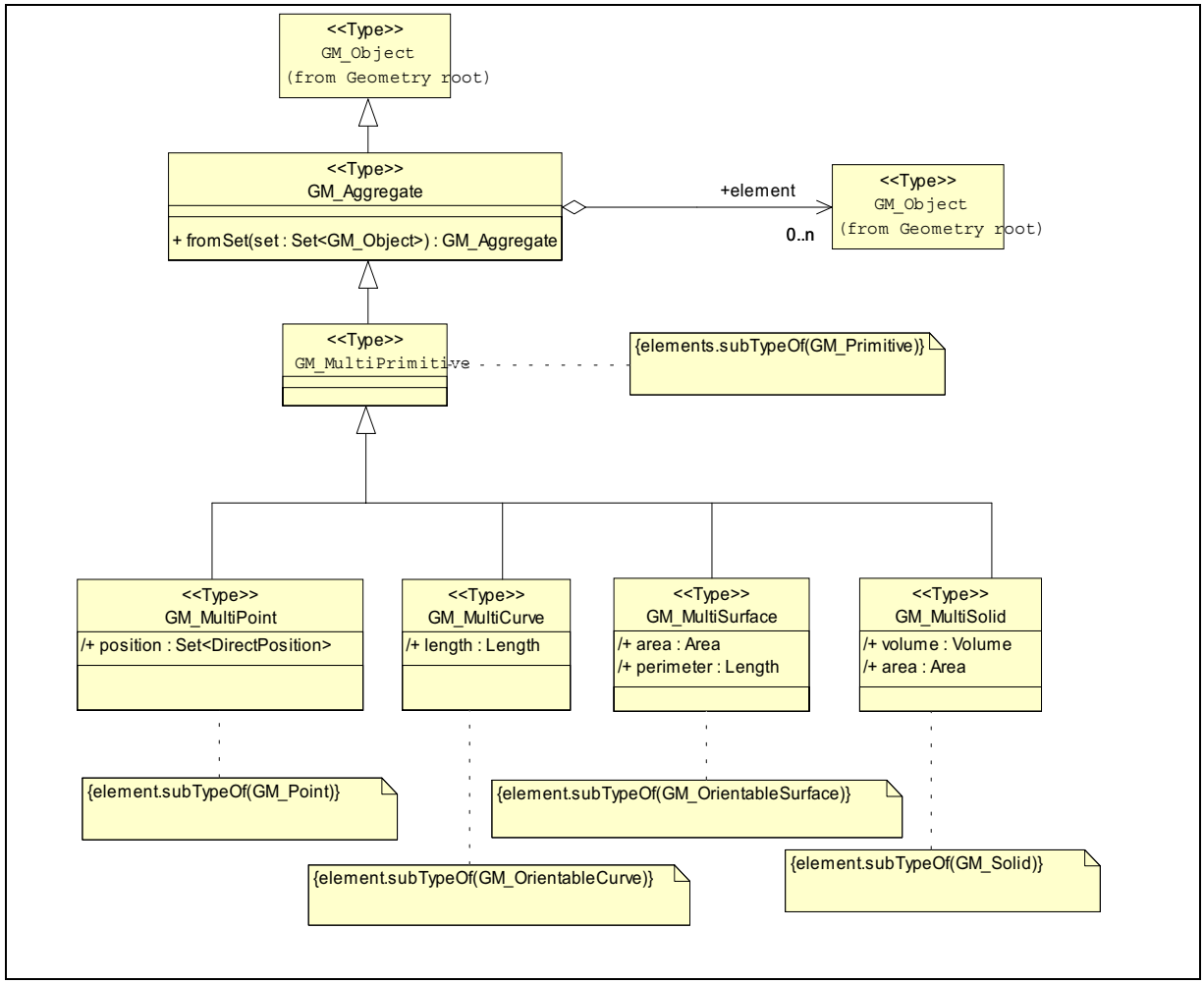


Figura 5 Il tipo GM_Aggregate come definito nello standard ISO 19107.

Non esistono nello standard ISO 19107 specifiche più precise per l'operazione di boundary delle classi GM_Complex e GM_Aggregate. Ciò si giustifica per il fatto che gli oggetti appartenenti a queste due classi possono avere a priori una struttura molto complessa e, in particolare, possono essere oggetti a dimensionalità mista, vale a dire, possono presentare ad esempio parti di dimensione 2 (poligoni) insieme a parti di dimensione 1 (linee) o di dimensione 0 (punti). Un esempio di oggetto della classe GM_Complex a dimensionalità mista viene mostrato in Figura 6. Si noti che tale oggetto secondo la definizione dell'operazione dimension() della classe GM_Object ha dimensione 2.

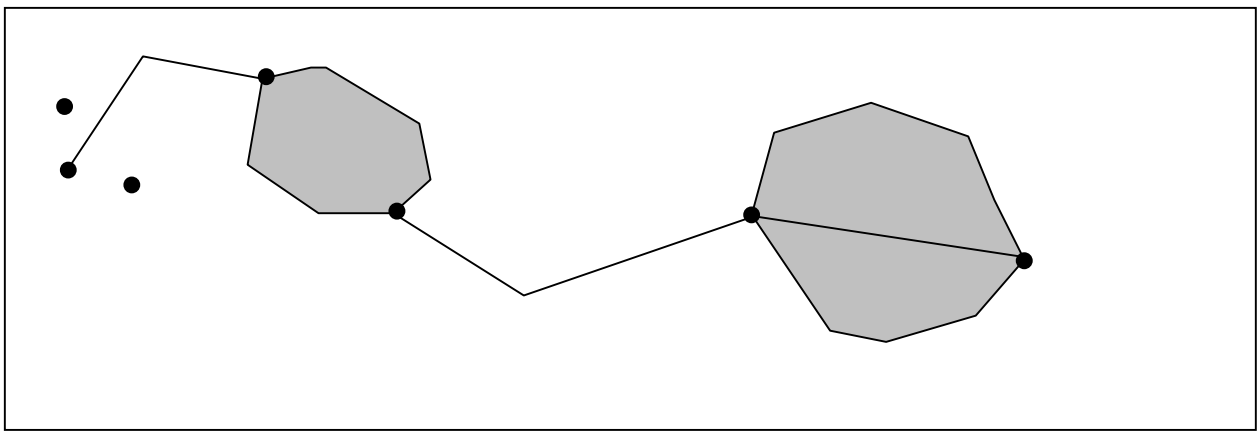


Figura 6 Un oggetto della classe GM_Complex a dimensionalità mista.

Per un oggetto come quello mostrato in Figura 6 esistono diversi modi di definire il boundary soddisfacendo sempre i vincoli espressi nello standard ISO 19107. In particolare le due seguenti interpretazioni sono quelle più vicine alla nozione intuitiva di boundary applicata ad un oggetto a dimensionalità mista:

1. poiché l'oggetto ha dimensione 2, costituiscono boundary tutte le linee (oggetti GM_Curve) non condivise da più poligoni (oggetti GM_Surface) (si considera ovviamente la chiusura delle linee, visto che il boundary è sempre un oggetto della classe GM_Complex).
2. il boundary dell'oggetto si ottiene dall'unione del boundary di tutti i suoi componenti, considerando solo quelli che non fanno parte del boundary di altri componenti.

Nella successiva Figura 7 si mostra il boundary dell'oggetto rappresentato in Figura 6 nelle due interpretazioni sopra descritte. Si noti che entrambe le situazioni sono compatibili con quanto specificato nello standard ISO 19107.

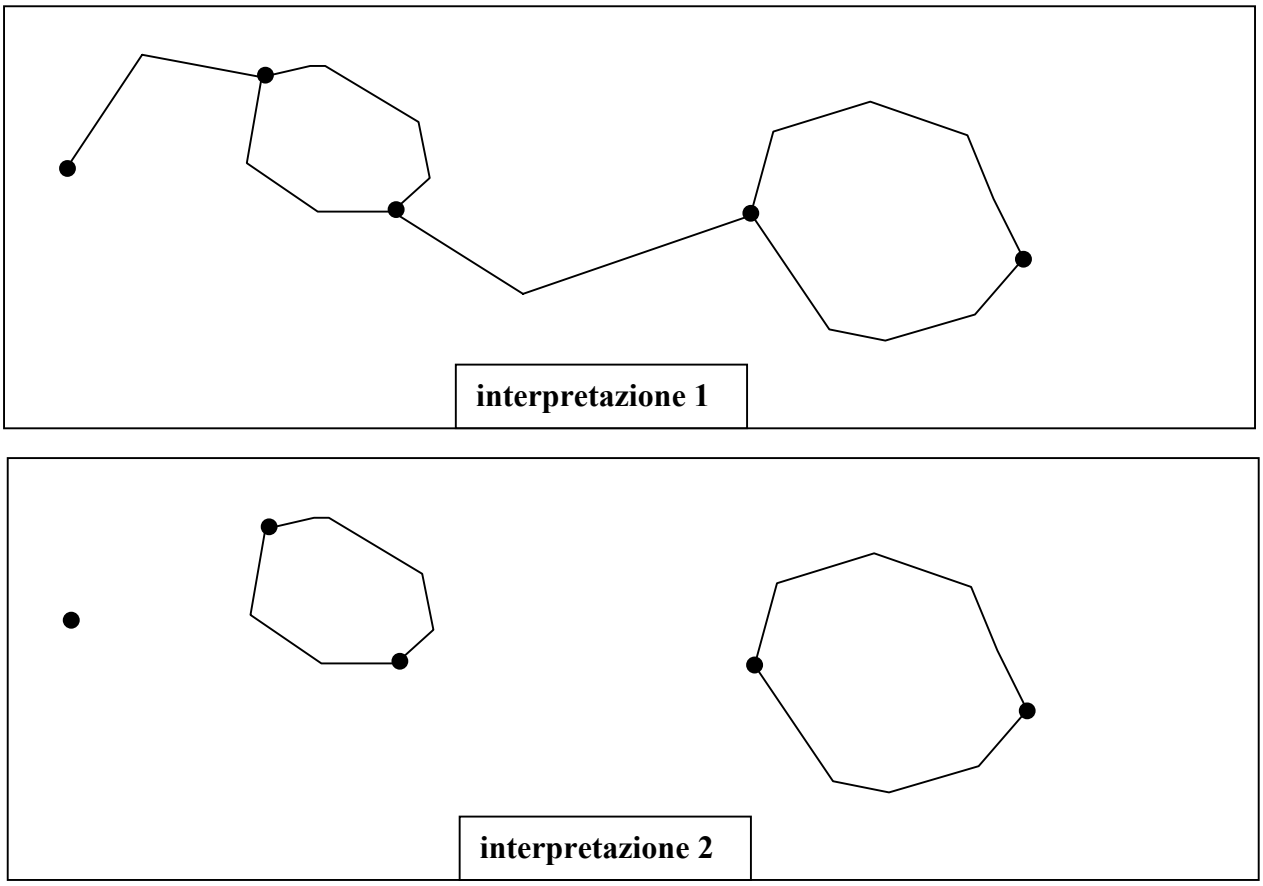


Figura 7 Diverse versioni del boundary di un oggetto a dimensionalità mista.

In GeoUML il boundary degli oggetti complessi e aggregati non viene definito. Ciò risulta in accordo anche con quanto indicato esplicitamente nello standard ISO 19107 - clause 8.1, dove introducendo le relazioni topologiche derivate si dice:

"of the following three classification techniques...What is to follow is only valid for point, curve, surface and solid objects. The theory for aggregate objects that are not homogeneous in dimension is not yet satisfactory enough to base a standard on."

Tuttavia, in GeoUML esistono due specializzazioni della classe GM_Complex, chiamate *GU_CXCurve2D* (*GU_CXCurve3D*) e *GU_CXSurface2D*, che hanno lo scopo di definire i sottoinsiemi della classe GM_Complex che sono omogenei rispetto alla dimensione 1 e 2 rispettivamente.

A tali classi specializzate in GeoUML si attribuisce una definizione precisa di boundary. Tale definizione non compromette la compatibilità con lo standard ISO 19107 visto che quanto richiesto dallo standard viene comunque soddisfatto.

Definizione 33. Boundary degli oggetti della classe *GU_CXCurve2D* **(***GU_CXSurface2D***)**

L'operazione boundary della classe GM_CXCurve2D (GM_CXSurface2D) è specificata in OCL nel seguente modo:

context *GM_CXCurve2D::boundary(): GM_ComplexBoundary*

post: *result.element =*

```
self.element→iterate(a: GM_Primitive, acc: Set = Set{} |
    if a.dimension() = self.dimension - 1 and
        not self.element→ exists(b: GM_Primitive,
            self.element→ exists(c: GM_Primitive,
                c<>b and
                b.boundary().element→ includes(a) and
                c.boundary().element→ includes(a)))
    then
        acc.union(a.closure().element)
    endif
```

Tale definizione indica che il boundary di un oggetto O della classe *GM_CXCurve2D* (o *GM_CXSurface2D*) è costituito da tutti gli oggetti GM_Point (o GM_Curve e loro boundary) elementi di O che non sono parte del boundary di altri due elementi distinti di O di tipo GM_Curve (o GM_Surface). Per alcuni esempi si veda la Figura 8.

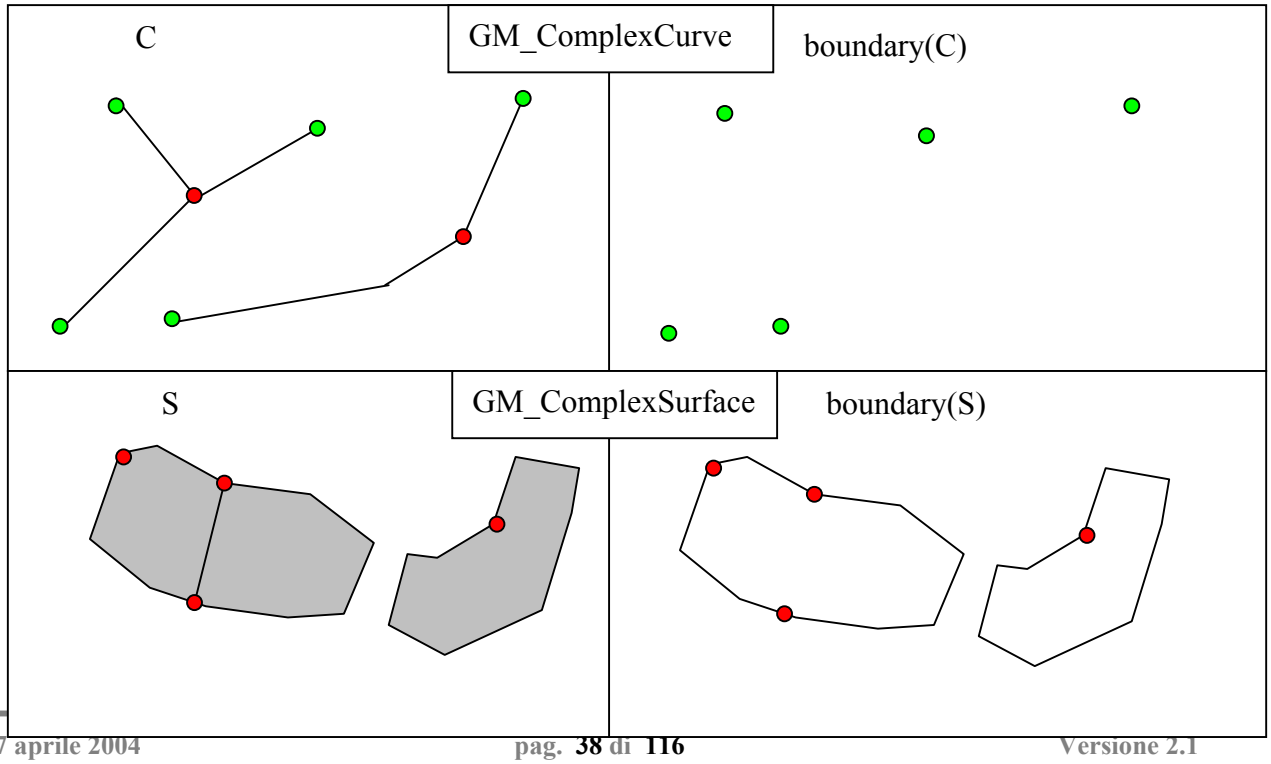


Figura 8 Esempi di boundary di oggetti GM_ComplexCurve e GM_ComplexSurface

2.6 Associazione (binaria)

Un'associazione rappresenta un legame logico tra due classi. Le istanze di una associazione rappresentano le coppie di oggetti appartenenti alle classi coinvolte nell'associazione. Per ogni classe partecipante ad una associazione si indicano: il ruolo che essa svolge nell'associazione e i vincoli di cardinalità che indicano a quante istanze dell'associazione può partecipare un'istanza di ogni classe.

Definizione 34. Associazione binaria

Un'associazione tra due classi C_1 e C_2 , $a(C_1, C_2)$, è una relazione binaria tra le due classi e quindi un'istanza di $a(C_1, C_2)$ ha come estensione un sottoinsieme del prodotto cartesiano $ext(C_1) \times ext(C_2)$:

$$ext(a(C_1, C_2)) \subseteq ext(C_1) \times ext(C_2)$$

Definizione 35. Ruolo di una classe in una associazione

Per le classi partecipanti ad una associazione è possibile specificare un ruolo che indica con quale funzione le istanze della classe partecipano all'associazione. Il ruolo consente di "navigare" l'associazione sulle istanze delle classi come fosse un attributo della classe stessa. Data l'associazione $a(r_1: C_1, r_2: C_2)$, dove r_1 e r_2 rappresentano i ruoli rispettivamente di C_1 e di C_2 in a, la scrittura:

$$c_1.r_2$$

con $c_1 \in ext(C_1)$, indica l'insieme delle istanze di C_2 collegate nella relazione a con l'istanza c_1 di C_1.

Definizione 36. Vincoli di cardinalità di una classe in una associazione

Per le classi partecipanti ad una associazione è possibile specificare un vincolo di cardinalità. Data un'associazione $a(C_1, C_2)$ il vincolo di cardinalità min..max specificato sulla classe $C_1(C_2)$ indica, fissata un'istanza dell'altra classe $C_2(C_1)$, quante istanze di associazione possono esistere.

[N.B. La definizione dei vincoli di cardinalità segue le regole del linguaggio UML ed è diversa da quella del modello Entità Relazione, usato per la specifica di schemi concettuali di basi di dati tradizionali]

La specifica completa di un'associazione prevede quindi l'indicazione dei ruoli e dei vincoli di cardinalità per tutte le classi partecipanti.

$$a(r_1[min_1..max_1]: C_1, r_2[min_2..max_2]: C_2)$$

Un'associazione può possedere attributi propri. In tal caso è necessario aggiungere la specifica di tali attributi in modo simile a quanto avviene per le classi.

Rappresentazione testuale (le regole fuori dal riquadro sono già state presentate in precedenza)

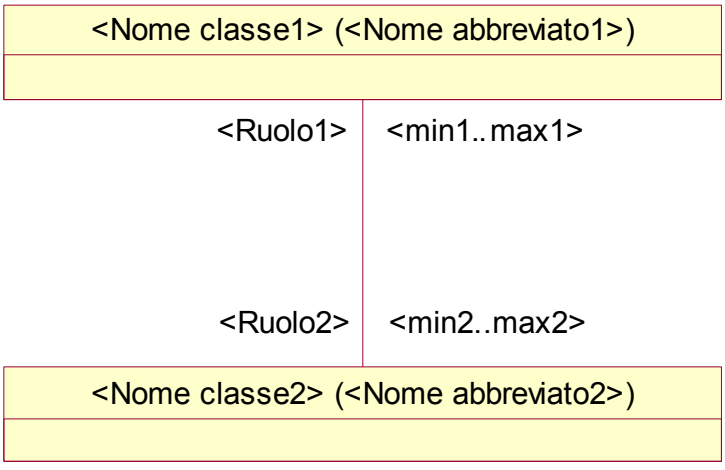
* <classe> → *classe* <identificatore classe> <proprietà della classe> **.**
<identificatore classe> → <identificatore> *(* <identificatore abbreviato> *)* | <identificatore>
<proprietà della classe> → *attributi:* <lista attributi> <ruoli> <vincoli>

<ruoli> → ε
<ruoli> → *; ruoli:* <lista ruoli>
<lista ruoli> → <ruolo>
<lista ruoli> → <ruolo> **;** <lista ruoli>
*<ruolo> → <ruolo classe associata> <cardinalità> *:* <classe associata>
inverso <ruolo inverso> <cardinalità>
*<ruolo> → <ruolo classe associata> <cardinalità> *:* <classe associata>
<ruolo classe associata> → <identificatore>
<ruolo inverso> → <identificatore>
<classe associata> → <identificatore>

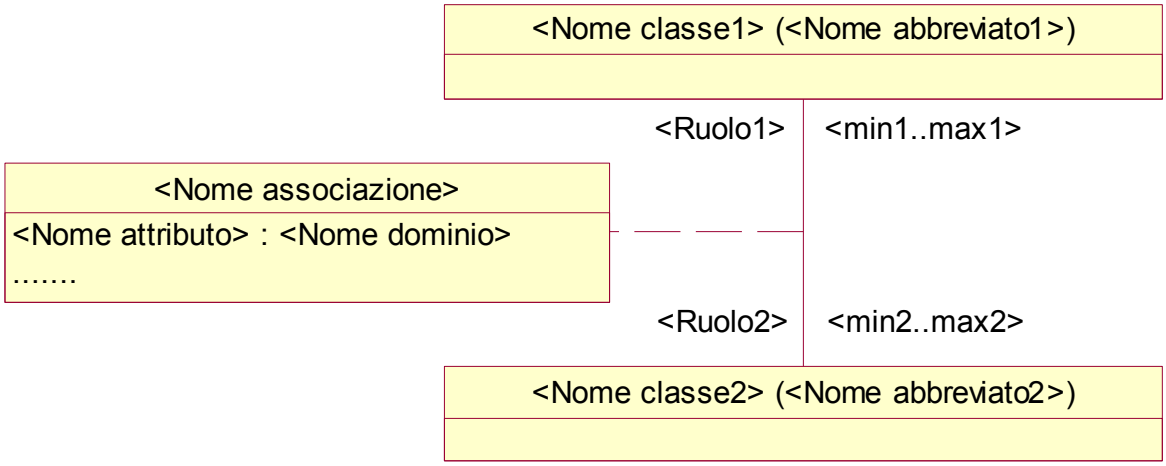
Nel caso in cui sull'associazione siano presenti attributi la rappresentazione testuale va completata con l'indicazione esplicita di tali attributi:

<associazione> → *associazione* <associazione> (<prima classe>, <seconda classe>)
<proprietà della associazione> **.**
<associazione> → <identificatore> **(** <identificatore abbreviato> **)** | <identificatore>
<proprietà della associazione> → *attributi :* <lista attributi alfa> <vincoli>
<prima classe> → <identificatore>
<seconda classe> → <identificatore>
<lista attributi alfa> → <attributo alfa>
<lista attributi alfa> → <attributo alfa> **;** <lista attributi alfa>

Rappresentazione grafica (senza attributi)



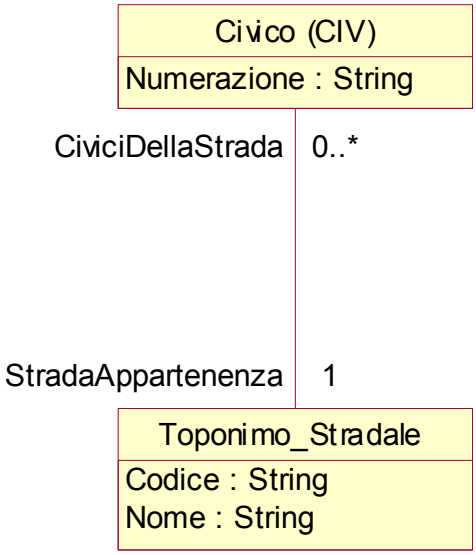
Rappresentazione grafica (con attributi)



Esempio

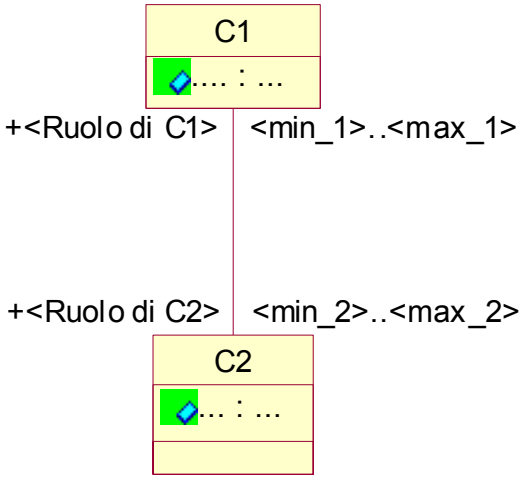
classe Civico (CIV)
attributi:
Numerazione: *String;*
ruoli:
StradaAppartenenza [1..1]: STR
inverso CiviciDellaStrada [0..*].

classe Toponimo stradale (STR)
attributi:
Codice: *String;*
Nome: *String*
ruoli: CiviciDellaStrada [0..*]:CIV
inverso StradaAppartenenza [1..1].

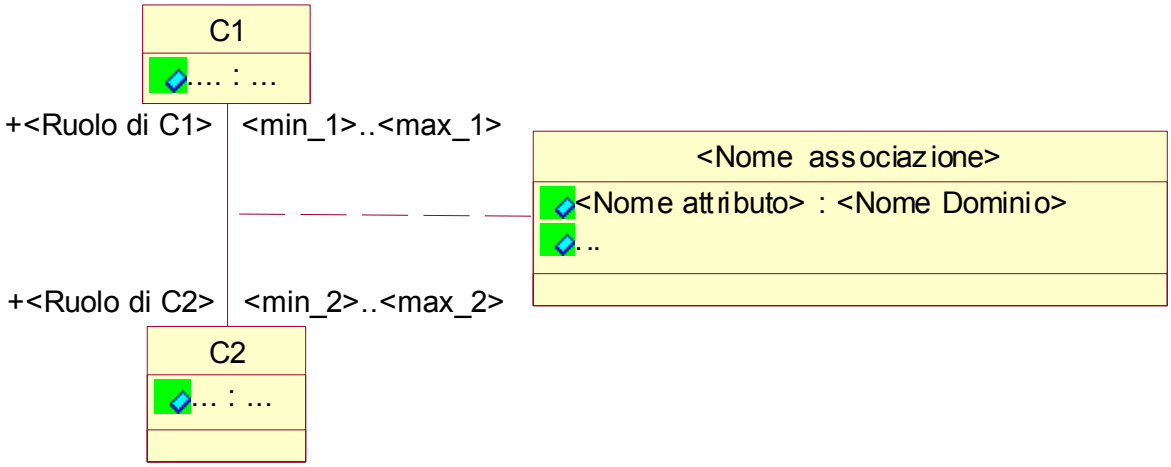


Traduzione in UML

Ogni associazione tra due classi C_1 e C_2 del modello concettuale si traduce in un'associazione ("association") del linguaggio UML con l'indicazione dei ruoli e degli eventuali vincoli di cardinalità come specificato nello schema concettuale.



Nel caso in cui l'associazione contenga attributi la traduzione è la seguente:



2.7 Gerarchia di ereditarietà tra le classi

Una gerarchia di ereditarietà rappresenta una relazione supertipo/sottotipo tra due classi, dove una classe (figlia) è sottotipo dell'altra classe (padre). La classe figlia eredita tutte le proprietà della classe padre. Le istanze della classe figlia sono istanze anche della classe padre.
Le gerarchie si possono combinare tra loro (un padre con più figli, figli di figli, ecc...).

Definizione 37. Gerarchia di ereditarietà

Una gerarchia di ereditarietà tra due classi C_{padre} e C_{figlia}, definisce un legame tra le classi con i seguenti effetti:

- *C_{figlia} eredita tutte le proprietà di C_{padre} (attributi, associazioni e vincoli): gli oggetti appartenenti a C_{figlia} avranno tutte le proprietà specificate in C_{figlia} e in aggiunta tutte le proprietà di C_{padre}.*
- *$ext(C_{figlia}) \subseteq ext(C_{padre})$.*

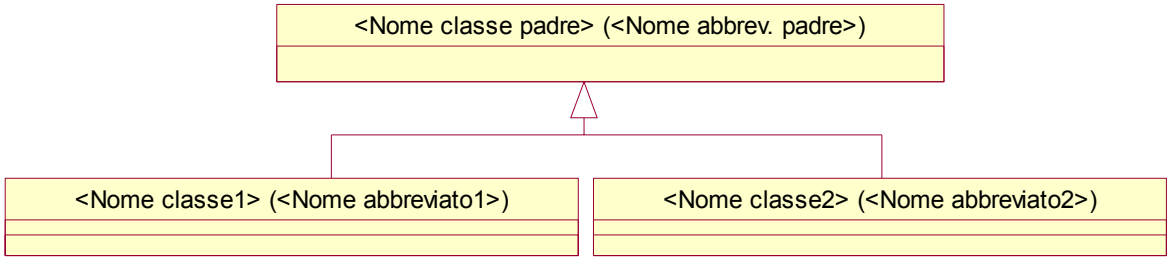
Rappresentazione testuale

* <classe> → *classe* <identificatore classe> <sottoclasse di> <proprietà della classe> **.**
<sottoclasse di> → *sottoclasse di* <identificatore altra classe> | ε
<identificatore altra classe> → <identificatore>

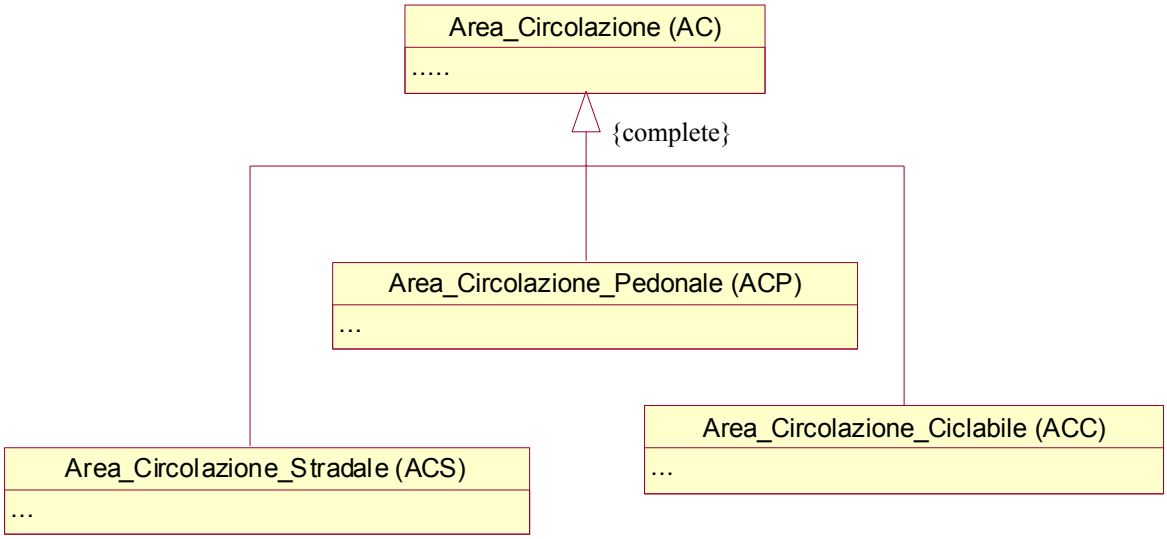
N.B.: si rappresenta il legame tra classe figlia e classe padre, mentre non si rappresenta il legame inverso. Se è necessario precisare che la gerarchia di ereditarietà è completa, vale a dire una istanza della classe padre appartiene ad almeno una classe figlia, allora va precisato con la seguente sitassi sulla classe padre.

<classe> → *classe* <identificatore classe> <classificazione completa> <sottoclasse di>
<proprietà della classe> **.**
<classificazione completa> → <vincoli classificazione> *(* <lista classi figlie> *)*
<vincoli classificazione> → <vincolo classif> | <vincolo classif 1> *,* <vincolo classif 2>
<vincolo classif> → *complete* | *incomplete* | *disjoint* | *overlapping*
<vincolo classif 1> → *complete* | *incomplete*
<vincolo classif 2> → *disjoint* | *overlapping*
<lista classi figlie> → <classe figlia>
<lista classi figlie> → <classe figlia> , <lista classi figlie>
<classe figlia> → <identificatore>

Rappresentazione grafica



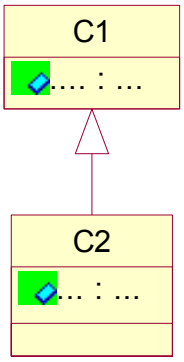
Esempio



classe Area_Circolazione (AC) *complete* (ACS, ACP, ACC)
attributi:... .
classe Area_Circolazione_Stradale (ACS) *sottoclasse di* AC
attributi:... .
classe Area_Circolazione_Pedonale (ACP) *sottoclasse di* AC
classe Area_Circolazione_Ciclabile (ACC) *sottoclasse di* AC

Traduzione in UML

Ogni gerarchia di ereditarietà tra due (o più) classi C_1 e C_2 (...C_n) di GeoUML si traduce direttamente in una gerarchia di ereditarietà ("generalization") del linguaggio UML.



2.8 Vincoli di integrità generici in OCL

Sia sulle classi che sulle associazioni di GeoUML è possibile definire vincoli di integrità. Tali vincoli stabiliscono l'insieme degli stati consistenti per le istanze delle classi o delle associazioni a cui sono legati. Il linguaggio adottato per la specifica dei vincoli è il linguaggio OCL (Object Constraint Language) di UML.

Definizione 38. Vincolo di integrità

Un vincolo di integrità definito su una classe C o su un'associazione A definisce una proprietà che deve essere sempre verificata su tutte le istanze della classe o dell'associazione. Tale proprietà viene espressa in OCL.

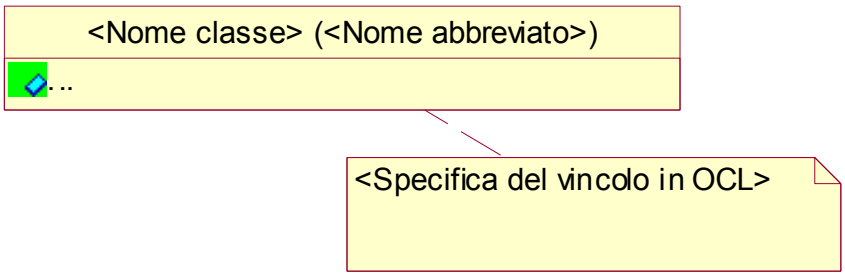
Rappresentazione testuale (le regole fuori dal riquadro sono già state presentate in precedenza)

<classe> → *classe* <identificatore classe> <classificazione completa> <sottoclasse di> <proprietà della classe> **.**

<proprietà della classe> → *attributi:* <lista attributi> <ruoli> <vincoli>

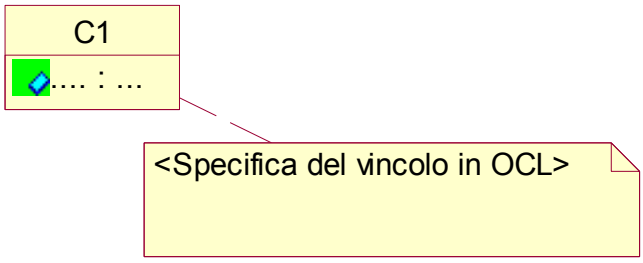
<vincoli> → ε
<vincoli> → *;* *vincoli:* <lista vincoli>
<lista vincoli> → <vincolo>
<lista vincoli> → <vincolo> *;* <lista vincoli>
<vincolo>→ <vincolo in OCL>
<vincolo in OCL> → "vedi sintassi OCL"

Rappresentazione grafica



Traduzione in UML

Ogni vincolo di integrità definito su una classe C1 (o associazione A) di GeoUML si traduce direttamente in un vincolo ("constraint") del linguaggio UML.



3 Costrutti derivati

3.1 Vincolo di chiave primaria

Tutte le classi sono dotate di un identificatore automatico interno (chiave surrogata), non "visibile" all'esterno della classe; tuttavia in molti casi è opportuno definire un identificatore esterno e "visibile", detto, secondo la terminologia del modello relazionale, chiave primaria.
Il vincolo di chiave primaria definisce un insieme di attributi e/o ruoli in associazioni che hanno la proprietà di identificare in modo univoco un oggetto istanza di una classe. Per quanto riguarda i ruoli viene richiesto che i vincoli di cardinalità posti sul ruolo siano [1..1]: vale a dire, da un oggetto della classe attraverso il ruolo si deve raggiungere uno e un solo oggetto dell'altra classe associata.
La scelta di attribuire o meno una chiave primaria a una classe deve essere valutata con attenzione, perché può avere rilevanti conseguenze implementative.
[N.B. Quando la chiave primaria è costituita da un unico attributo numerico generato dal sistema spesso è detta UUID (Universal Unique Identifier)]

Definizione 39. Chiave primaria

La chiave primaria di una classe C è un insieme $a_1, ..., a_k$ di attributi e/o ruoli in associazioni definiti sulla classe C tali che:

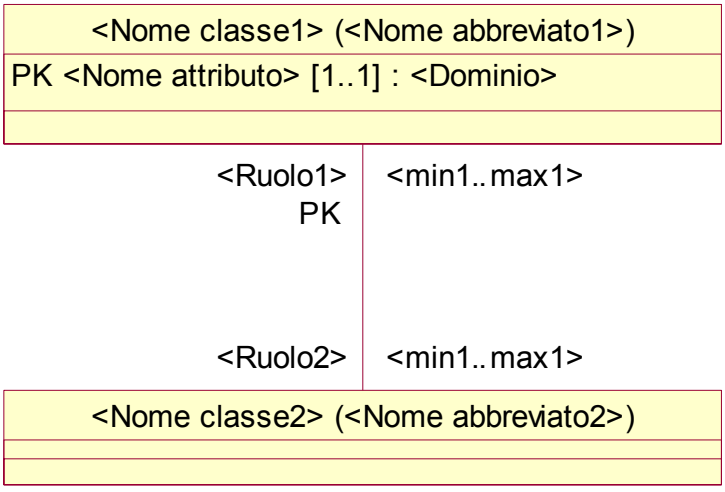
$$\forall o_i,o_j \in ext(C): o_i \neq o_j \Rightarrow (o_i.a_1 \neq o_j.a_1 \vee ... \vee o_i.a_k \neq o_j.a_k).$$

I simboli di disuguaglianza sono da interpretare come disuguaglianza sul valore per gli attributi con domini di base e come negazione di identità tra oggetti negli altri casi. Il vincolo di cardinalità posto sui ruoli coinvolti nella chiave deve essere [1..1].

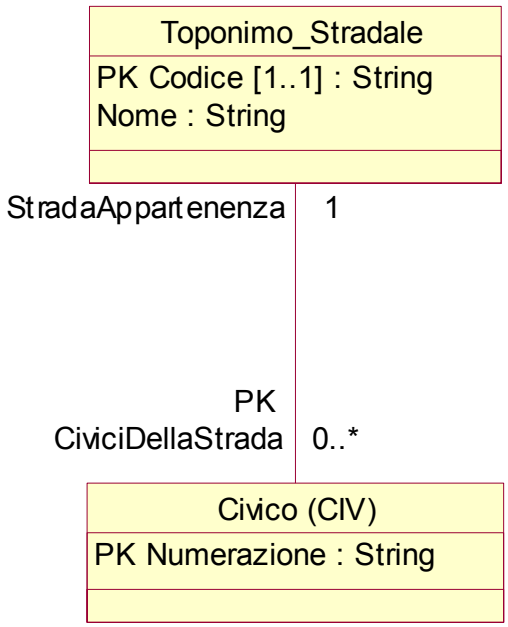
Rappresentazione testuale

<attributo alfa> → <chiave> <nome attributo> <cardinalità> *:* <dominio attributo>
<chiave> → ε | *PK*
<ruolo> → <chiave> <ruolo classe associata> <cardinalità> *:* <classe associata>
inverso <ruolo inverso> <cardinalità>
<ruolo> → <chiave> <ruolo classe associata> <cardinalità> *:* <classe associata>

Rappresentazione grafica



Esempio
In questo esempio si suppone che un toponimo stradale sia identificato dal proprio codice mentre un civico sia identificato dal numero nell'ambito della associazione con la sua strada di appartenenza.



Traduzione nei costrutti base del modello

Ogni vincolo di chiave primaria che coinvolge gli attributi (o ruoli) k_1, ..., k_n definito su una classe C di GeoUML si traduce direttamente in OCL nel seguente modo:

context *C*
inv: *C.allInstances→forAll* (c_1: *C* |
C.allInstances→forAll (c_2:*C* | c_1<>c_2 *implies*
(($c_1.k_1$ <>$c_2.k_1$) or... or ($c_1.k_n$ <>$c_2.k_n$))))

3.2 Attributo enumerato gerarchico

In alcuni casi è necessario rappresentare attributi i cui valori sono definiti attraverso una classificazione gerarchica. Ciò accade ad esempio nel caso della classificazione della fauna e della flora, ma anche nell'ambito geografico, ne è un esempio la classificazione dell'uso del suolo o delle rocce.

Tale tipo di attributo non può essere rappresentato con un attributo normale di GeoUML, ma richiede l'introduzione di un nuovo costrutto derivato che viene chiamato: attributo enumerato gerarchico. Si tratta di un attributo che assume un valore strutturato su due o più livelli ed è molto simile al valore rappresentato dal record con varianti presente nei linguaggi di programmazione. Al primo livello l'attributo enumerato gerarchico è uguale ad un attributo enumerato, quindi può assumere un valore compreso in una data lista di valori (dominio enumerato). Inoltre, a differenza dell'attributo enumerato, che non ha una ulteriore strutturazione, l'attributo enumerato gerarchico presenta in corrispondenza di uno o più dei valori enumerati v_i del primo livello una ulteriore struttura. Tale struttura è costituita da un certo insieme di attributi che assumono valori significativi quando il valore del primo livello è v_i. Tale struttura è ricorsiva e può quindi ripetersi più volte.

Definizione 40. Attributo enumerato gerarchico

Un attributo enumerato gerarchico di dominio D associato ad una classe C è una funzione così definita:

$$a: ext(C) \rightarrow D$$

Il dominio D è definito da una gerarchia di classi (G, A), dove $G = \{C_0, ..., C_n\}$, $n \geq 1$ *e* $A \subseteq C \times C$, *con i seguenti vincoli:*

- $(C_i, C_j) \in A$ *se* C_i *è padre di* C_j.
- *la classe* C_0 *è la classe radice della gerarchia e ha nome C_a;*
- *ogni classe non foglia* C_i *della gerarchia ha un attributo enumerato* a_i;
- *ogni classe non foglia* C_i *della gerarchia ha un insieme di classi figlie che corrispondono ad un sottoinsieme* $\{v_1, ..., v_k\}$ *dei valori presenti nel dominio del suo attributo enumerato: per ogni valore* v_j *esiste una classe figlia* $C_i\_v_j$.
- *ogni classe non foglia* C_i *della gerarchia soddisfa il seguente vincolo:*

$$\forall o_i \in C_i: ((c_i \in C_i\_v_j) \Rightarrow (o_i.a_i = v_j))$$

Rappresentazione testuale (le regole fuori dal riquadro sono già state presentate in precedenza)

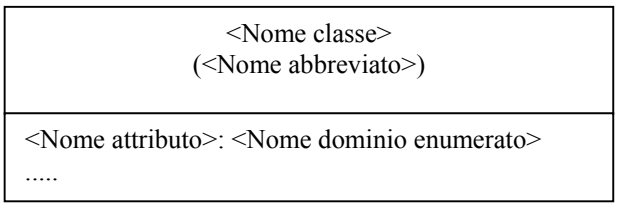
<attributo alfa> → <chiave> <nome attributo> <cardinalità> **:** <dominio attributo>
<dominio alfa> → <identificatore> | <dominio enumerato> | <u>String</u> | <u>Integer</u> | <u>Real</u>
<dominio enumerato> → *(* <lista valori> *)*
<lista valori> → <valore>
<lista valori> → <valore> **,** <lista valori>

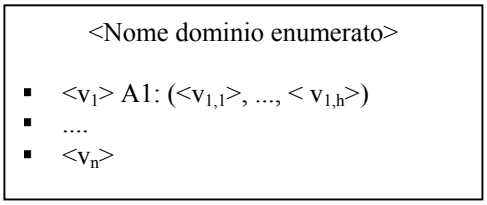
<valore> → <identificatore> | <valore cond>
<valore cond> → <identificatore> <sep> <lista cond opzionale>
<lista cond opzionale> → <attributo_cond> *: (* <lista valori> *)*
<lista cond opzionale> → <attributo_cond> *: (* <lista valori> *)* <lista cond opzionale>
<attributo_cond> → <identificatore>
<sep> → "spazi, tab e caratteri speciali per indicare fine riga"

Si noti che non esiste nella grammatica un non terminale per il dominio enumerato gerarchico. Esso viene rappresentato come una variante del dominio enumerato

Ad esempio
A [0..1]: (v_1 **A1:(**$v_{1,1}$ **B1:(**$v_{1,1,1}$, $v_{1,1,2}$**),** $v_{1,2}$ **,** $v_{1,3}$**)** , v_2 $\mathbf{A_{n-1}}$**:(**$v_{2,1}$**,** $v_{2,2}$**)** $\mathbf{A_n}$**:(**$v_{2,3}$**,** $v_{2,4}$**),** v_3, v_4**)**

Rappresentazione grafica (solo con dominio dichiarato)





Traduzione nei costrutti base del modello

Un attributo enumerato gerarchico *a* di una classe *C* può essere rappresentato usando i costrutti base di GeoUML nel seguente modo:

- Viene aggiunto un attributo normale *a* di tipo *C_a* alla classe *C*.
- Viene aggiunta allo schema una gerarchia di classi con radice nella classe *C_a*. La classe *C_a* contiene un solo attributo a_0 (di nome "*valore*") e ha una serie di classi figlie *C_a_v1*, ..., *C_a_vk* che corrispondono ai valori v_i specificati al primo livello della gerarchia per i quali è presente una ulteriore strutturazione. Ogni classe figlia *C_a_vi*, se non è una classe foglia della gerarchia, contiene certamente un attributo a_i (il cui nome è definito nella rappresentazione testuale) ed è a sua volta padre di una gerarchia con i vincoli sopra descritti.

Su ogni classe non foglia C_i della gerarchia con sottoclassi C_1, ..., C_n è inoltre specificato il seguente vincolo di integrità OCL:

context $\boldsymbol{C_i}$

inv: $C_i.allinstances \rightarrow forall(o_i: C_i \mid$

$$((c_i \in C_i\_v_1) \Rightarrow (o_i.tipo\_v_i = v_1) \text{ and } ... \text{ and } (c_i \in C_i\_v_n) \Rightarrow (o_i.\ tipo\_v_i = v_n))$$

3.3 Strato topologico: classe con un attributo geometrico (complesso) monoistanza

In molti casi è necessario rappresentare nello schema concettuale un oggetto che rappresenti l'insieme complessivo di tutte le istanze di una certa categoria di informazioni. Tale esigenza può nascere ad esempio per l'insieme delle linee di trasporto di un certo tipo (la rete stradale, la rete ferroviaria, ecc...) oppure per le classificazioni del territorio (uso del suolo, limiti amministrativi, ecc...). Tale rappresentazione di insieme non è soltanto l'unione di tutte le istanze di un certo tipo, ma esplicita in una unica struttura le relazioni topologiche esistenti tra le istanze. Nei sistemi che gestiscono basi di dati geografiche, tali rappresentazioni si chiamano spesso strati (o layer). Per indicare la presenza esplicita di una struttura topologica per la geometria delle istanze coinvolte, in GeoUML tali strati prendono il nome di strati topologici.

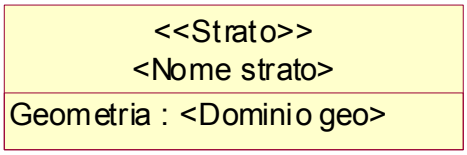
Uno strato topologico viene rappresentato attraverso una classe con i seguenti vincoli:

- la classe ha un unica istanza: vale a dire, nella base di dati ci sarà al massimo uno e un sol oggetto appartenente a tale classe (classe monoistanza);
- la classe ha uno e un solo attributo geometrico, detto *geometria*, di tipo: GU_Complex2D o GU_Complex3D o GU_CXSurface2D o GU_CXCurve2D o GU_CXCurve3D o *GU_CNCurve2D* o *GU_CNCurve3D* o *GU_CXRing2D* o *GU_CXRing3D*.

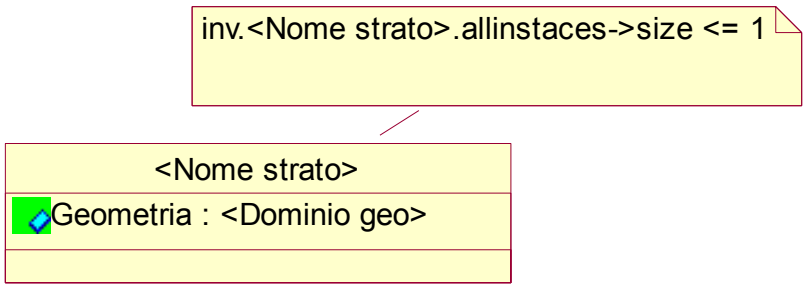
Rappresentazione testuale

<strato topologico> → *strato* <id strato> <proprietà strato>.
<proprietà strato> → *geometria:* <dominio strato> <vincoli>
<id strato> → <identificatore> (<identificatore abbreviato>) | <identificatore>
<dominio strato> → *GU_Complex2D* | *GU_Complex3D* | *GU_CXCurve2D* | *GU_CXCurve3D* | *GU_CXSurface2D* | *GU_CNCurve2D* | *GU_CNCurve3D* | *GU_CXRing2D* | *GU_CXRing3D*

Rappresentazione grafica



Traduzione nei costrutti base del modello



3.4 Vincoli di integrità basati su relazioni topologiche

La specifica di schemi concettuali richiede molto frequentemente di indicare vincoli di integrità di natura spaziale, vale a dire, condizioni che devono essere sempre vere e che fanno riferimento alle proprietà spaziali degli oggetti, in particolare alle relazioni topologiche tra oggetti. Le relazioni topologiche costituiscono la formalizzazione delle relazioni spaziali intuitive che vengono comunemente usate nell'analisi di una carta geografica da parte dell'utente umano.

La forma dei vincoli topologici spesso prevede la presenza di un *quantificatore esistenziale* (∃), vale a dire, il vincolo richiede l'*esistenza* di una certa relazione spaziale tra le istanze di due classi dello schema. Più precisamente il vincolo richiede che data un'istanza c di una classe C *esista* un'istanza c' in un'altra classe C' tale che valga una certa relazione spaziale tra c e c'. In taluni casi non è richiesto che la relazione sia soddisfatta da un'unica istanza c', ma dall'unione di un insieme di istanze appartenenti alla classe C'. Altre varianti sono legate all'uso del quantificatore universale invece dell'esistenziale e all'esistenza di associazioni tra le classi coinvolte nel vincolo.

La definizione dei vincoli topologici richiede quindi di specificare due tipi di elementi:

1. le relazioni spaziali utilizzate nel vincoli
2. la struttura logica del vincolo

Per quanto riguarda la definizione delle relazioni spaziali, a differenza di quanto fatto nei capitoli precedenti, dove la formalizzazione è stata data come traduzione in OCL, qui essa viene fornita utilizzando un metodo diverso, più semplice e intuitivo (paragrafo 3.4.1), mentre la traduzione in OCL dell'espressione delle relazioni è rimandata al capitolo 3.5.

Nei paragrafi di questo capitolo successivi al 3.4.1 vengono definite e formalizzate in OCL le varie forme che i vincoli topologici possono assumere; dato che le relazioni topologiche definite al capitolo 3.4.1 non sono in OCL, nella definizione formale dei vincoli in OCL si fa riferimento a una funzione

check (relazione topologica)

la cui definizione nei termini dello Spatial Schema è rinviata al capitolo successivo (3.5).

3.4.1 Le relazioni topologiche utilizzabili nell'espressione di vincoli

Per definire i vincoli topologici è necessario individuare un'insieme di relazioni topologiche di riferimento. Ciò non è semplice in quanto valori geometrici di tipo diverso danno vita a relazioni topologiche diverse, e non è banale individuare un insieme di relazioni valido per tutti i tipi. Tuttavia esiste un'insieme di relazioni topologiche, proposto in letteratura da Clementini et alter, che risulta applicabile a tutti i valori geometrici semplici e connessi di interesse GeoUML (punto, linea e poligono).

La definizione formale delle relazioni topologiche di Clementini è la seguente:

Definizione 41. Relazioni topologiche di riferimento REL_{topo}

Dati due valori geometrici A e B di cui sia possibile individuare una parte interna o interior (A° e B°) e di conseguenza di una frontiera o boundary, possono essere definite le seguenti relazioni topologiche:

$(A \text{ Disjoint } B) \equiv_{def} (A \cap B = \varnothing)$
$(A \text{ Touch } B) \equiv_{def} (A° \cap B° = \varnothing) \text{ and } (A \cap B \neq \varnothing)$
$(A \text{ In } B) \equiv_{def} (A \cap B = A) \text{ and } (A° \cap B° \neq \varnothing) \text{ and } (A \cap B \neq B)$
$(A \text{ Contains } B) \equiv_{def} (B \text{ In } A)$
$(A \text{ Equal } B) \equiv_{def} (A \cap B = A) \text{ and } (A \cap B = B)$
$(A \text{ Overlap } B) \equiv_{def} (A° \cap B° \neq \varnothing) \text{ and } (A \cap B \neq A) \text{ and } (A \cap B \neq B)$

dove $\cap$ indica l'intersezione insiemistica tra punti

Una ulteriore relazione topologica molto utile, non primaria, ma derivabile dalle altre, è la seguente:

Definizione 14. Relazione topologica Intersects

$(A \text{ Intersects } B) \equiv_{def} (A \cap B \neq \varnothing) \text{ and not } (A \text{ Equal } B)$

Si può dimostrare che la relazione Intersects corrisponde alla disgiunzione delle relazioni Touch, Overlap e Cross:

$(A \text{ Intersects } B) \Leftrightarrow (A \text{ Touch } B) \text{ or } (A \text{ Overlap } B) \text{ or } (A \text{ In } B) \text{ or } (A \text{ Contains } B))$

Questa definizione formale delle relazioni topologiche corrisponde alle situazioni geometriche mostrate in Figura 9. Come si nota, nella figura sono riportati solo oggetti con geometria semplice (punti, linee e poligoni), appartenenti quindi alle classi dello Spatial Schema: GM_Point, GM_Curve e GM_Surface rispettivamente. Per tali classi la funzione boundary è precisamente definita e anche le relazioni topologiche sono ben definite e di facile interpretazione.

Un'ulteriore relazione che in taluni casi può essere interessante specificare è quella che, in presenza di una relazione di Overlap tra due linee, caratterizza il sottocaso in cui l'intersezione tra gli interior sia costituita da un insieme finito di punti isolati.

Definizione 14bis. Relazione topologica Cross

$(A \text{ Cross } B) \equiv_{def} (A \text{ Overlap } B) \text{ and } (dim(A)=1) \text{ and } (dim(B)=1) \text{ and } (dim(A° \cap B°)= 0)$

dove dim(X) è una funzione che vale 0 se X è un punto, 1 se X è una linea e 2 se X è un poligono.

Le relazioni topologiche dell'insieme REL_{topo} sono applicabili in realtà anche ad oggetti con geometria qualsiasi, purché per ogni oggetto sia univocamente definito il suo boundary e quindi la sua parte interna (interior).

Non tutte le classi di GeoUML soddisfano tale condizione; elenchiamo di seguito le classi che la soddisfano:

- le sottoclassi di GM_Point (e quindi sottoclassi di GM_Primitive): *GU_Point2D* e *GU_Point3D*;
- le sottoclassi di GM_Composite in quanto sottoclasse di GM_Primitive: infatti poiché ogni oggetto della classe GM_Composite è anche oggetto di GM_Primitive ha un boundary, e

quindi una parte interna, ben definiti: *GU_CPCurve2D*, *GU_CPCurve3D*, *GU_Ring2D*, *GU_Ring3D* e *GU_CPSurface2D.*

- le sottoclassi di GM_Complex che sono dimensionalmente omogenee: *GU_CXPoint2D*, *GU_CXPoint3D*, *GU_CXCurve2D*, *GU_CXCurve3D*, *GU_CXRing2D*, *GU_CXRing3D* e *GU_CXSurface2D*.

Le classi dello Spatial Schema per le quali lo standard ISO 19107 non specifica una definizione precisa di boundary sono: GM_Aggregate e GM_Complex. Di conseguenza le classi di GeoUML per la quali non viene definito il boundary sono: *GU_Complex2D*, *GU_Complex3D* e tutte le classi che ereditano da GM_Aggregate. Pertanto, nell'applicazionec di vincoli topologici agli oggetti di queste classi è necessario rispettare la seguente restrizione:

Restrizione
I vincoli topologici basati sulle relazioni Touch, Overlap e Cross non possono essere applicati agli oggetti della classe GM_Aggregate e GM_Complex a dimensionalità mista e quindi in particolare alle classi: *GU_Complex2D, GU_Complex3D, GU_Aggregate2D e GU_Aggregate3D*. E' possibile invece applicare agli oggetti di queste classi i vincoli topologici basati sulle relazioni: Equal, Disjoint e Intersects.

Presentiamo ora la definizione dei vincoli esistenziali basati sulle relazioni topologiche dell'insieme REL_{topo}. Tale definizione sarà data in modo incrementale, vale a dire, vengono indicate diverse varianti del vincolo esistenziale che possono essere combinate tra loro. Nei casi reali è tuttavia assai raro l'uso contemporaneo di tutte le varianti possibili.

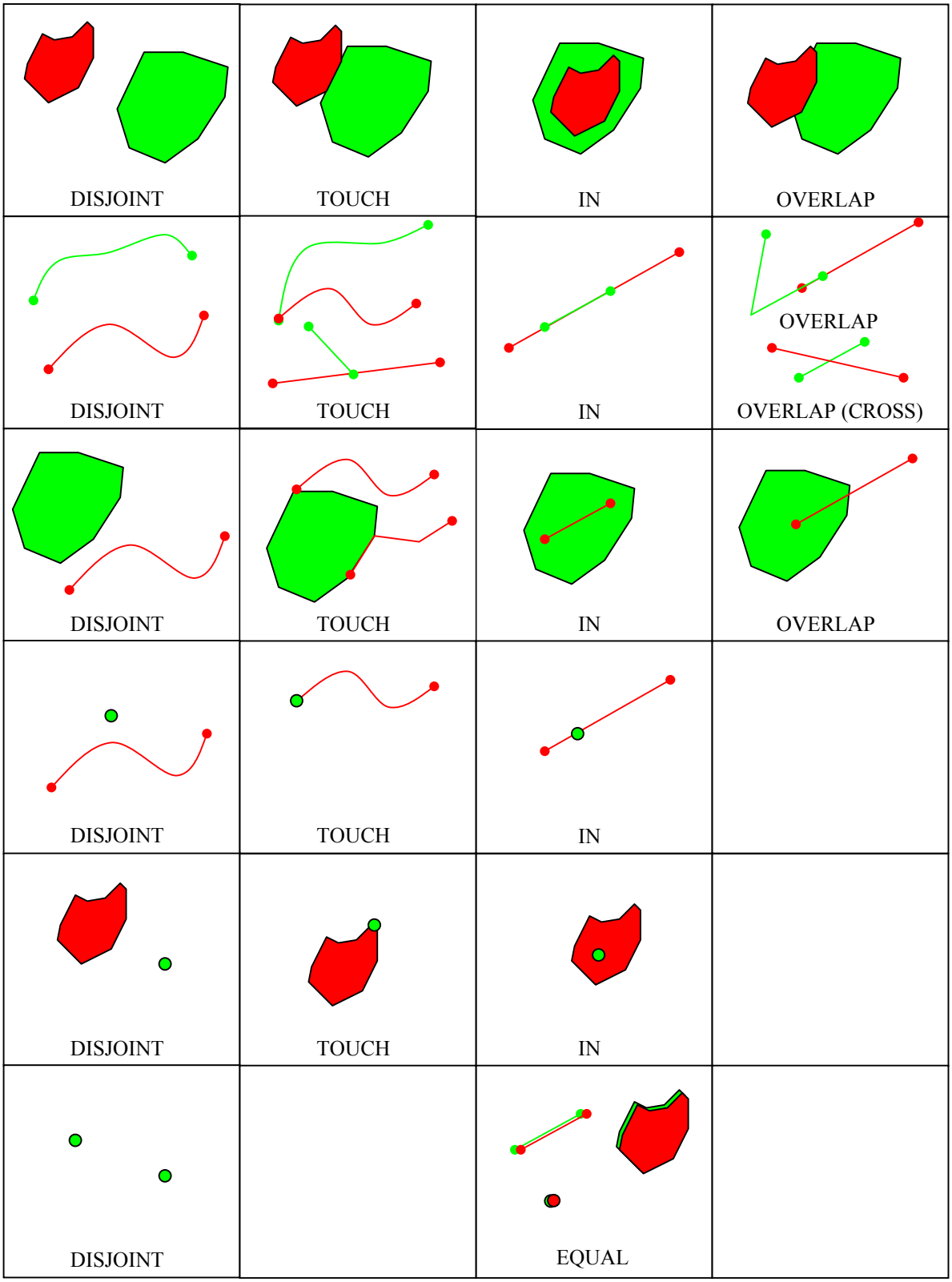


Figura 9 Rappresentazione geometrica delle relazioni topologiche di Clementini per oggetti semplici e connessi. In ogni colonna viene rappresentato un tipo di relazione e in ogni riga una coppia di tipi di oggetti. Nell'ultima colonna nella riga che rappresenta le relazioni tra linee e tra poligoni e linee è stata aggiunta la relazione CROSS. Infine nell'ultima riga, nella colonna che contiene esempi della relazione IN, sono stati riportati esempi della relazione EQUAL.

Si noti inoltre che la definizione di vincoli topologici che coinvolgono le classi *GU_CPSurfaceB3D* e *GU_CXSurfaceB3D* richiedono una specializzazione delle definizioni date di seguito e verranno illustrati nel paragrafo 3.4.9.

3.4.2 Vincolo topologico esistenziale di base

La forma più semplice di vincolo esistenziale basato su relazioni topologiche è quello che si applica a classi contenenti un unico attributo geometrico; esso richiede per ogni istanza c della classe vincolata l'esistenza di almeno una istanza c' della classe vincolante che si trovi in una certa relazione topologica con c oppure che si trovi in una relazione topologica appartenente ad un certo gruppo di relazioni (disgiunzione di relazioni).

Definizione 42. Vincolo topologico esistenziale di base

Date due classi X e Y contenenti almeno un attributo geometrico ciascuna, rispettivamente g e f, si dice vincolo topologico esistenziale da X verso Y, basato sulla disgiunzione di relazioni $rel_1, ..., rel_k$ *il seguente vincolo OCL:*

$Topo_{\exists}(X, g, \{rel_1, ..., rel_k\}, Y, f) \equiv$

context *X*

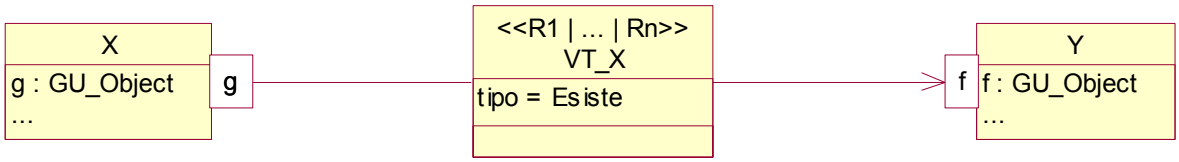
inv: $Y.allinstances \rightarrow exists(a: Y \mid check(self.g, \{rel_1, ..., rel_k\}, a.f)$

Rappresentazione testuale del vincolo

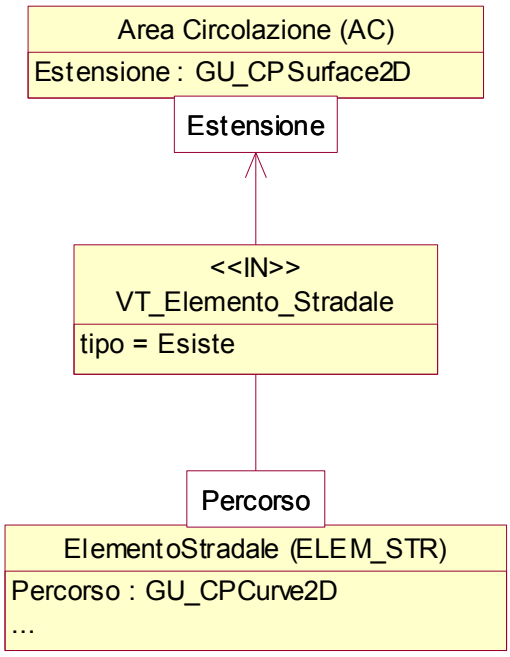
* <vincolo topologico> → *Vtopo* <classe vincolata> <lista relazioni> *esiste* <classe vincolante>
* <classe vincolata> → <identificatore>.<identificatore_attr_geo>
* <classe vincolante> → <identificatore>.<identificatore_attr_geo>
<identificatore_attr_geo> → <identificatore>
<lista relazioni> → <relazione topo>
<lista relazioni> → <relazione topo> , <lista relazioni>
<relazione topo> → *DJ* | *TC* | *IN* | *CT* | *EQ* | *CR* | *OV*

Rappresentazione grafica del vincolo

La rappresentazione grafica del vincolo topologico esistenziale ha la seguente forma:



Esempio


Area Circolazione (AC)
Estensione : GU_CPSurface2D
Estensione
<<IN>>
VT_Elemento_Stradale
tipo = Esiste
Percorso
ElementoStradale (ELEM_STR)
Percorso : GU_CPCurve2D
...

3.4.3 Vincolo topologico esistenziale con selezioni

Una prima variante del vincolo è quella che introduce la possibilità di selezionare le istanze della classe X coinvolte nel vincolo e/o di selezionare le istanze delle classe Y che possono essere scelte per soddisfare il vincolo (a causa della presenza del quantificatore esistenziale). Ciò significa che il vincolo consente di specificare delle condizioni di selezione sia su X che su Y.

Definizione 43. Vincolo topologico esistenziale con selezioni

Date due classi X e Y contenenti almeno un attributo geometrico ciascuna, rispettivamente g e f, si dice vincolo topologico esistenziale da X verso Y con selezioni, basato sulla disgiunzione di relazioni rel_1, ...,rel_k il seguente vincolo OCL:

$Topo_{\exists}(X, \sigma_1(x), g, \{rel_1, ...,rel_k\}, Y, \sigma_2(x, y), f) \equiv$

context *X*

inv: *$\sigma_1(self)$ implies (Y.allinstances→ exists(a: Y | $\sigma_2(self, a)$ and*
check(self.g, $\{rel_1, ...,rel_k\}$, a.f)

dove le espressioni $\sigma_1(self)$ e $\sigma_2(self, a)$ indicano le espressioni che si ottengono sostituendo self al posto di x in $\sigma_1(x)$ e self e a al posto di x e y rispettivamente in $\sigma_2(x,y)$ rispettivamente.

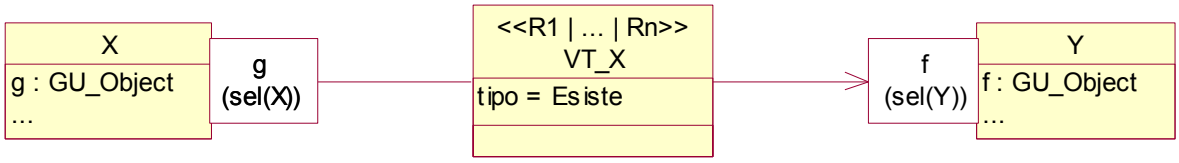
Nota: $\sigma_2(self, a)$ utilizza entrambe le variabili self e a solo quando il predicato è un predicato di join, altrimenti self non viene usato.

Rappresentazione testuale del vincolo (le regole fuori dal riquadro sono già state presentate in precedenza)

* <vincolo topologico> → *Vtopo* <classe vincolata> <selezioneX> <lista relazioni>
esiste <classe vincolante> <selezioneYX>
<selezioneX> → ε | *(* <qualsiasi espressione boolean in OCL sugli attributi della classe vincolata X> *)*
<selezioneYX> → ε | *(* <qualsiasi espressione boolean in OCL sugli attributi della classe vincolante Y e eventualmente della classe vincolata X> *)*

* <classe vincolata> → <identificatore>.<identificatore_attr_geo>
* <classe vincolante> → <identificatore>.<identificatore_attr_geo>
<identificatore_attr_geo> → <identificatore>
<lista relazioni> → <relazione topo>
<lista relazioni> → <relazione topo> , <lista relazioni>
<relazione topo> → *DJ* | *TC* | *IN* | *CT* | *EQ* | *CR* | *OV*

Rappresentazione grafica del vincolo



Si noti che nell'espressione delle selezioni è possibile usare l'attributo OID, che si riferisce all'identificatore univoco dell'oggetto.

3.4.4 Vincolo topologico esistenziale sulla frontiera o sulla proiezione

Una seconda variante prevede invece la possibilità di applicare la relazione topologica alla frontiera (o boundary) dell'oggetto. In tal modo è possibile esprimere, ad esempio, il contenimento di un oggetto nel boundary di un altro. L'uso del boudary invece che dell'oggetto intero nelle relazioni produce una nuova variante del vincolo topologico esistenziale. La definizione è indipendente dalla specifica classe dello Spatial Schema usata come tipo per gli attributi geometrici, in quanto la funzione boudary è disponibile su tutte le classi, essendo una funzione della classe GM_Object.

Un'ulteriore variante consente di applicare la relazione topologica alla proiezione dell'oggetto nello spazio 2D. Tale proiezione si ottiene attraverso la funzione planar() che è stata aggiunta alla classe astratta *GU_Object* ed è quindi disponibile su tutte le classi di GeoUML. L'applicazione di tale funzione ovviamente è significativa solo per le classi che rappresentano oggetti dello spazio 3D.

Definizione 44. Vincolo topologico esistenziale sulla frontiera o sulla proiezione

Date due classi X e Y contenenti almeno un attributo geometrico ciascuna, rispettivamente g e f, si dice vincolo topologico esistenziale sulla frontiera (o proiezione) da X verso Y, basato sulla disgiunzione di relazioni rel_1, ...,rel_k il seguente vincolo OCL:

$Topo_{\exists}^{B-}(X, g, \{rel_1, ...,rel_k\}, Y, f) \equiv$

context *X*

inv: $Y.allinstances \rightarrow exists(a: Y \mid check(self.g.boundary(), \{rel_1, ...,rel_k\}, a.f)$

$Topo_{\exists}^{P-}(X, g, \{rel_1, ...,rel_k\}, Y, f) \equiv$

context *X*

inv: $Y.allinstances \rightarrow exists(a: Y \mid check(self.g.planar(), \{rel_1, ...,rel_k\}, a.f)$

La versione duale del vincolo è quella che presenta l'applicazione della funzione boundary (o della funzione planar) sugli oggetti della classe Y. L'indicazione degli attributi geometrici ai quali applicare la funzione boundary (o planar) è riportata ad apice della segnatura del vincolo. La stringa "B-" ("P-") indica l'applicazione della funzione boundary (planar) all'attributo della prima classe, mentre la stringa "-B" ("-P") indica l'applicazione della funzione alla seconda classe. Infine è possibile richiedere l'applicazione della funzione boundary (planar) su entrambe gli attributi con la stringa "BB" ("PP").

Rappresentazione testuale del vincolo (le regole fuori dal riquadro sono già state presentate in precedenza)

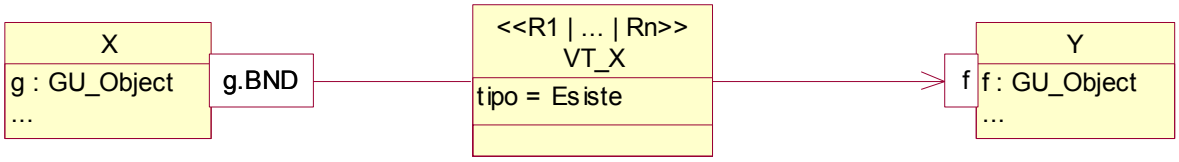
* <vincolo topologico> → *Vtopo* <classe vincolata> <selezioneX> <boundary>
<lista relazioni> *esiste*
<classe vincolante> <selezioneYX> <boundary>
<boundary> → ε | *.bnd* | *.pln* | *.bnd.pln* | *.pln.bnd*

* <classe vincolata> → <identificatore>.<identificatore_attr_geo>
* <classe vincolante> → <identificatore>.<identificatore_attr_geo>
<identificatore_attr_geo> → <identificatore>
<lista relazioni> → <relazione topo>
<lista relazioni> → <relazione topo> , <lista relazioni>
<relazione topo> → *DJ* | *TC* | *IN* | *CT* | *EQ* | *CR* | *OV*
<selezioneX> → ε | *(* <qualsiasi espressione boolean in OCL sugli attributi della classe vincolata X> *)*

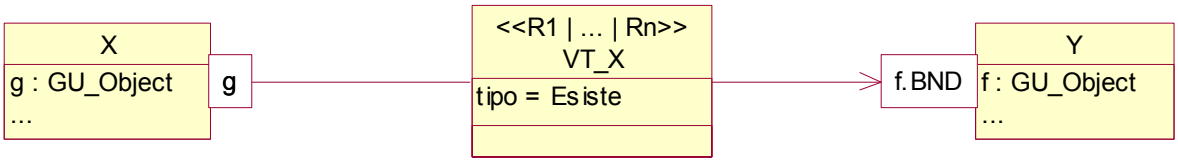
<selezioneYX> → ε | (<qualsiasi espressione boolean in OCL sugli attributi della classe vincolante Y e eventualmente della classe vincolata X>)

Rappresentazione grafica del vincolo

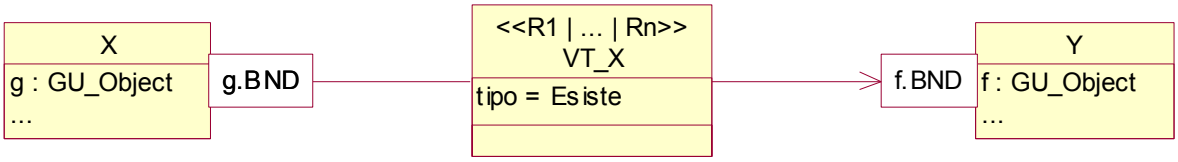
Graficamente l'applicazione della funzione boundary (planar) viene indicata con l'inserimento del suffisso ".BND" "PLN" posto sugli attributi geometrici dei quali si voglia considerare la frontiera (o la proiezione). Ad esempio se l'attributo di cui si vuole considerare la frontiera è quello della classe X allora la rappresentazione grafica sarà la seguente:



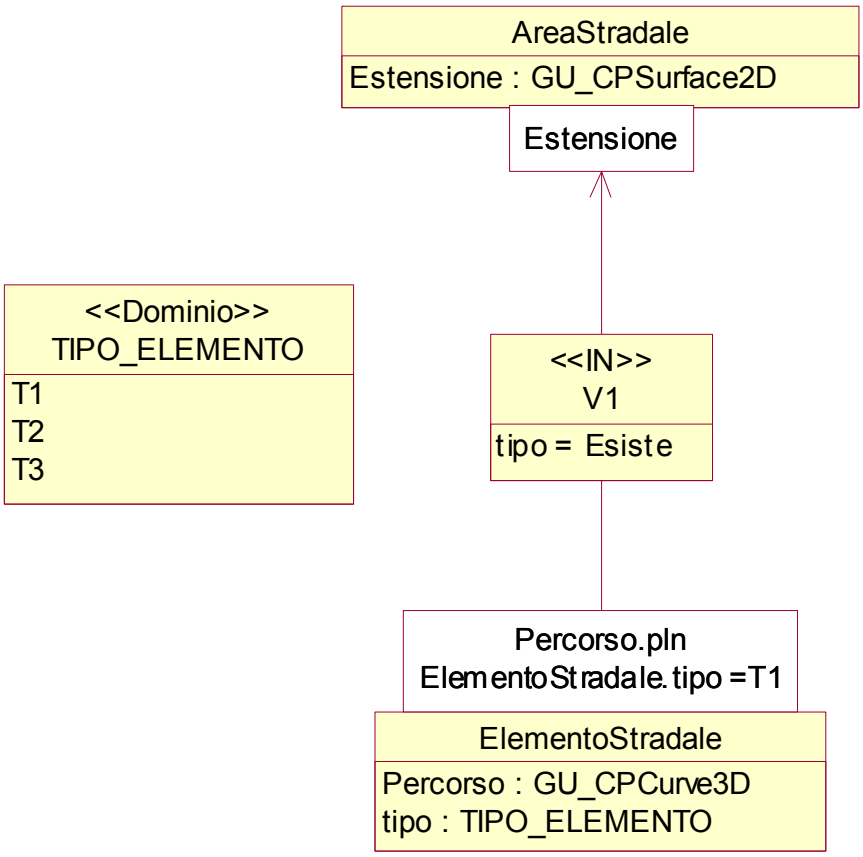
Se invece l'attributo di cui si vuole considerare la frontiera è quello della classe Y, allora la rappresentazione grafica sarà la seguente:



E' infine possibile anche la presenza su entrambe gli estremi del suffisso ".BND". Tale situazione indica che la frontiera viene richiesta su entrambe gli attributi.



Esempio


AreaStradale
Estensione : GU_CPSurface2D
Estensione
<<Dominio>>
TIPO_ELEMENTO
T1
T2
T3
<<IN>>
V1
tipo = Esiste
Percorso.pln
ElementoStradale.tipo =T1
ElementoStradale
Percorso : GU_CPCurve3D
tipo : TIPO_ELEMENTO

3.4.5 Vincolo topologico con unione

L'ultima versione del vincolo topologico di base è quella che considera l'unione degli oggetti che rappresentano le istanze dell'attributo geometrico della classe vincolante (chiamata qui solitamente classe Y). Ciò significa che la relazione topologica viene verificata non rispetto all'attributo geometrico di un'istanza di Y, ma rispetto all'oggetto che si ottiene dall'unione degli oggetti che rappresentano l'attributo geometrico di tutte le istanze di Y. Anche in questo caso, la funzione che calcola l'unione è indipendente dalla classe geometrica in quanto risulta disponibile nella classe padre astratta GU_Object.

Definizione 45. Vincolo topologico con unione

Date due classi X e Y contenenti almeno un attributo geometrico ciascuna, rispettivamente g e f, si dice vincolo topologico con unione da X verso Y, basato sulla disgiunzione di relazioni rel_1, ...,rel_k il seguente vincolo OCL:

$Topo_{Union}(X, g, \{rel_1, ...,rel_k\}, Y, f) \equiv$

context *X*

inv: *check(self.g, {rel_1, ...,rel_k}, Y.allinstances→*
*iterate (a: Y, acc: GM_Object = NULL | acc.*union *(a.f)))*

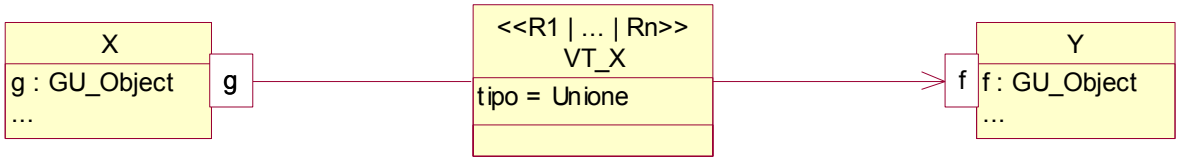
Rappresentazione testuale del vincolo (le regole fuori dal riquadro sono già state presentate in precedenza)

* <vincolo topologico> → *Vtopo* <classe vincolata> <selezioneX> <boundary>
<lista relazioni> *union*
<classe vincolante> <selezioneYX> <boundary>

* <classe vincolata> → <identificatore>.<identificatore_attr_geo>
* <classe vincolante> → <identificatore>.<identificatore_attr_geo>
<identificatore_attr_geo> → <identificatore>
<lista relazioni> → <relazione topo>
<lista relazioni> → <relazione topo> , <lista relazioni>
<relazione topo> → *DJ* | *TC* | *IN* | *CT* | *EQ* | *CR* | *OV*
<selezioneX> → ε | *(* <qualsiasi espressione boolean in OCL sugli attributi della classe vincolata X> *)*
<selezioneYX> → ε | *(* <qualsiasi espressione boolean in OCL sugli attributi della classe vincolante Y e eventualmente della classe vincolata X> *)*
<boundary> → ε | *.bnd* | *.pln* | *.bnd.pln* | *.pln.bnd*

Rappresentazione grafica del vincolo

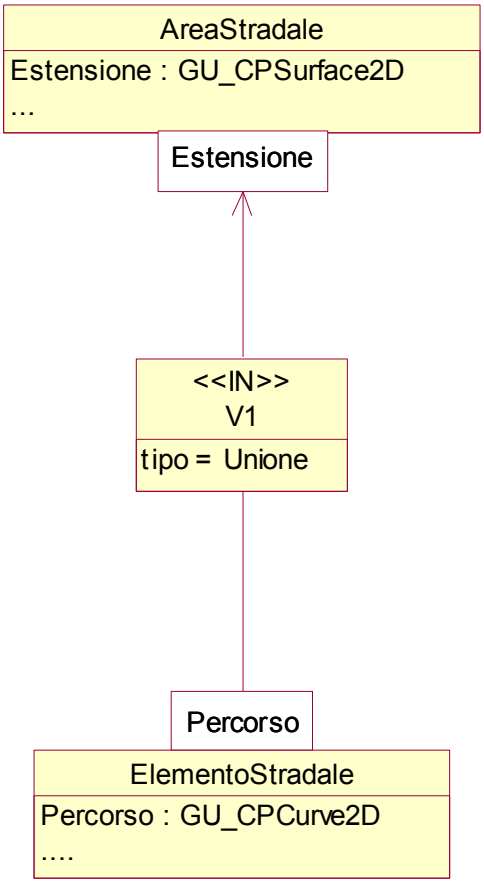
Graficamente l'unione sulle istanze dell'entità vincolante (Y) viene visualizzata indicando come tipo del vincolo topologico il valore "Unione".



[N.B. questo vincolo NON richiede il quantificatore esistenziale nella formulazione che è stata fornita; si poteva usare tale quantificatore formulandolo come: esiste un sottoinsieme

degli oggetti di Y che soddisfa il vincolo; per questo motivo è stato inserito nella categoria dei vincoli esistenziali]

Esempio



3.4.6 Vincolo topologico universale

Sostituendo il quantificatore esistenziale con quello universale otteniamo una nuova versione del vincolo topologico, che richiede di soddisfare una condizione "più forte" rispetto al vincolo esistenziale. Infatti, la soddisfazione del vincolo topologico con quantificatore universale richiede che la relazione topologica (o disgiunzione di relazioni topologiche) sia presente tra l'oggetto vincolato e tutti gli oggetti della classe vincolante.

Definizione 46. Vincolo topologico universale

Date due classi X e Y contenenti almeno un attributo geometrico ciascuna, rispettivamente g e f, si dice vincolo topologico universale da X verso Y, basato sulla disgiunzione di relazioni $rel_1, ..., rel_k$ *il seguente vincolo OCL:*

$Topo_{\forall}(X, g, \{rel_1, ..., rel_k\}, Y, f) \equiv$

context *X*

inv: *Y.allinstances→forall(a: Y | check(self.g, {*$rel_1, ..., rel_k$*}, a.f) or self=a)*

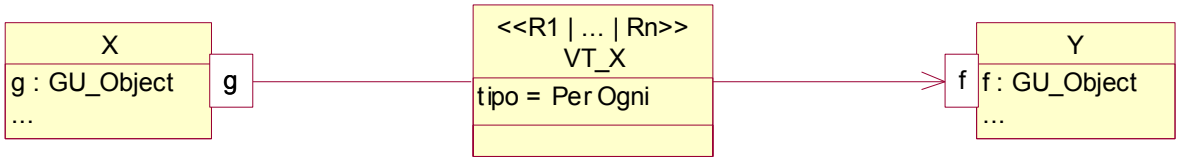
Rappresentazione testuale del vincolo (le regole fuori dal riquadro sono già state presentate in precedenza)

* <vincolo topologico> → *Vtopo* <classe vincolata> <selezioneX> <boundary>
<lista relazioni> *perOgni*
<classe vincolante> <selezioneYX> <boundary>

* <classe vincolata> → <identificatore>.<identificatore_attr_geo>
* <classe vincolante> → <identificatore>.<identificatore_attr_geo>
<identificatore_attr_geo> → <identificatore>
<lista relazioni> → <relazione topo>
<lista relazioni> → <relazione topo> , <lista relazioni>
<relazione topo> → *DJ* | *TC* | *IN* | *CT* | *EQ* | *CR* | *OV*
<selezioneX> → ε | *(* <qualsiasi espressione boolean in OCL sugli attributi della classe vincolata X> *)*
<selezioneYX> → ε | *(* <qualsiasi espressione boolean in OCL sugli attributi della classe vincolante Y e eventualmente della classe vincolata X> *)*
<boundary> → ε | *.bnd* | *.pln* | *.bnd.pln* | *.pln.bnd*

Rappresentazione grafica del vincolo

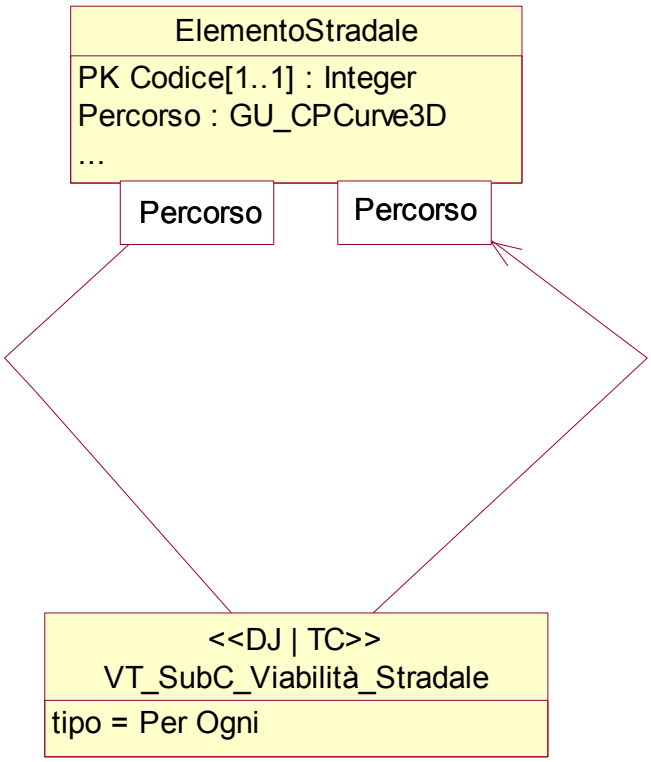
La presenza del quantificatore universale si indica graficamente viene visualizzata indicando come tipo del vincolo topologico il valore “Unione”. La direzione della freccia è quella dall’entità vincolata a quella vincolante (cioè la direzione nella quale si leggono le relazioni spaziali)



Anche per il vincolo topologico universale esistono le varianti presentate per il vincolo topologico esistenziale, vale a dire: la versione grafica con nomi espliciti degli attributi geometrici, la versione con selezione e la versione che si riferisce ai boundary degli attributi geometrici. La versione del vincolo topologico esistenziale, che fa riferimento all’unione degli oggetti che rappresentano

l'attributo geometrico delle istanze dell'entità vincolante, non contiene quantificatori e quindi non presenta una versione duale con il quantificatore universale.

Esempio



3.4.7 Vincolo topologico collegato ad una associazione

E' possibile che in uno schema concettuale sia necessario specificare un vincolo di integrità basato su relazioni topologiche, dove si richieda di considerare, come istanze della classe vincolante, solo gli oggetti che sono legati all'oggetto da vincolare attraverso un'associazione specificata nello schema. In tal caso la definizione di vincolo topologico esistenziale data nel paragrafo 3.4.2 (Definizione 42) deve essere specializzata per questo caso particolare, come segue.

Definizione 47. Vincolo topologico esistenziale con associazione

Siano date due classi X e Y contenenti almeno un attributo geometrico ciascuna, rispettivamente g e f, e tra le quali esista un'associazione Z, dove il ruolo di Y sia r. Si dice vincolo topologico esistenziale da X verso Y, basato sulla disgiunzione di relazioni rel_1, ...,rel_k e agganciato all'associazione Z, il seguente vincolo OCL:

$Topo_{\exists}(X, g, r, \{rel_1, ...,rel_k\}, Y, f) \equiv$

context *X*

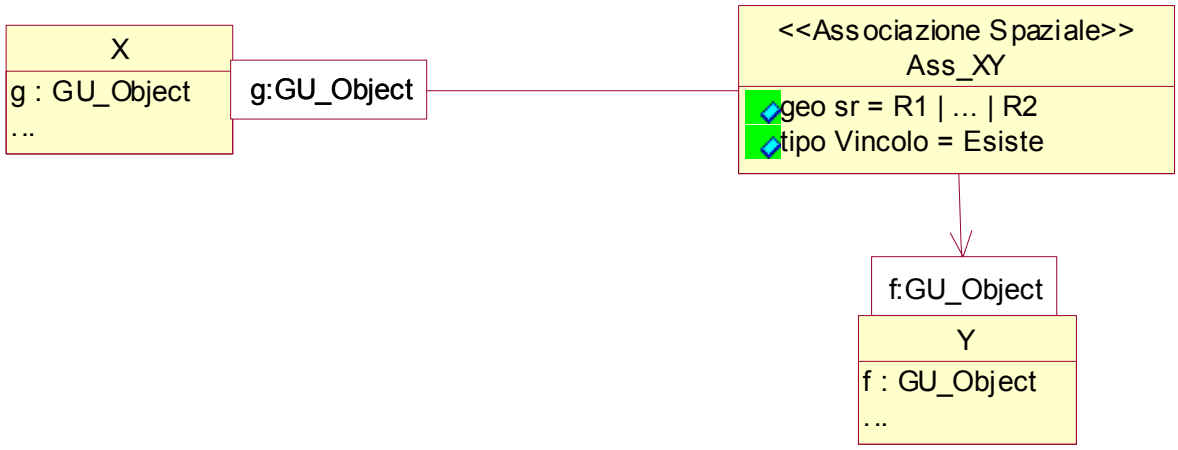
inv: $self.r \rightarrow exists(a: Y \mid check(self.g, \{rel_1, ...,rel_k\}, a.f))$

Rappresentazione testuale del vincolo (le regole fuori dal riquadro sono già state presentate in precedenza)

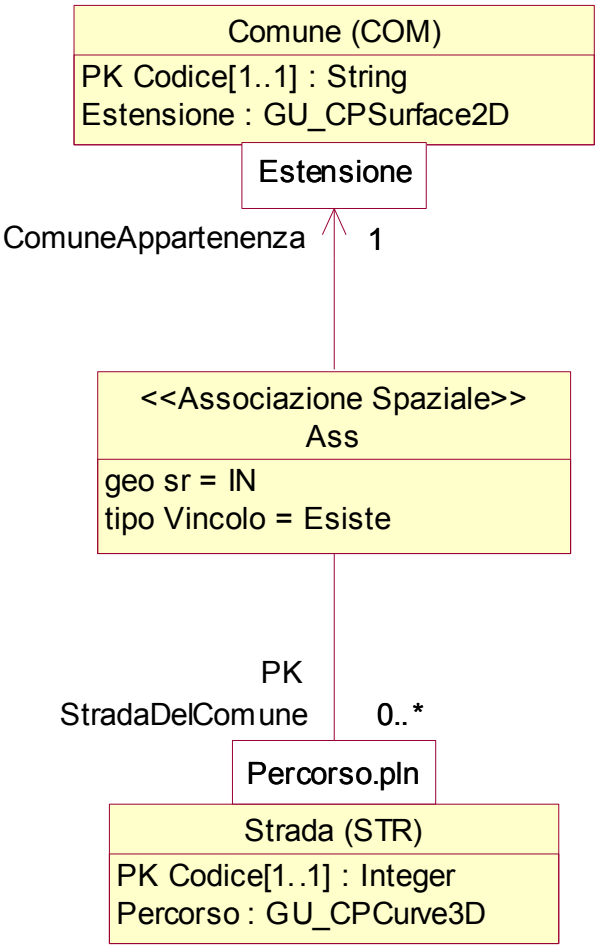
* <vincolo topologico> → *Vtopo* <classe vincolata> <selezioneX> <boundary>
<lista relazioni> <tipo vincolo>
<classe vincolante> <selezioneYX> <boundary>
* <classe vincolante> → <identificatore>.<identificatore_attr_geo> |
<identificatore>.<id ruolo>.<identificatore_attr_geo>
<tipo vincolo> → *esiste* | *union* | *perOgni*
<id ruolo> → <identificatore> | <identificatore>::<identificatore>

* <classe vincolata> → <identificatore>.<identificatore_attr_geo>
<identificatore_attr_geo> → <identificatore>
<lista relazioni> → <relazione topo>
<lista relazioni> → <relazione topo> , <lista relazioni>
<relazione topo> → *DJ* | *TC* | *IN* | *CT* | *EQ* | *CR* | *OV*
<selezioneX> → ε | *(* <qualsiasi espressione boolean in OCL sugli attributi della classe vincolata X> *)*
<selezioneYX> → ε | *(* <qualsiasi espressione boolean in OCL sugli attributi della classe vincolante Y e eventualmente della classe vincolata X> *)*
<boundary> → ε | *.bnd* | *.pln* | *.bnd.pln* | *.pln.bnd*

Rappresentazione grafica del vincolo



Esempio



3.4.8 Disgiunzione di vincoli topologici

E' possibile che in uno schema concettuale sia necessario specificare su una classe una disgiunzione di vincoli topologici esistenziali. Ciò accade ad esempio quando si richiede che una certa relazione topologica sia soddisfatta da un oggetto x rispetto ad un oggetto y_1 di una classe Y_1 oppure rispetto ad un oggetto y_2 di una classe Y_2.

La rappresentazione in OCL di una disgiunzione di vincoli topologici esistenziali si ottiene facilmente combinando in una espressione booleana attraverso l'operatore OR le rappresentazioni dei singoli vincoli così come definite nei paragrafi precedenti.

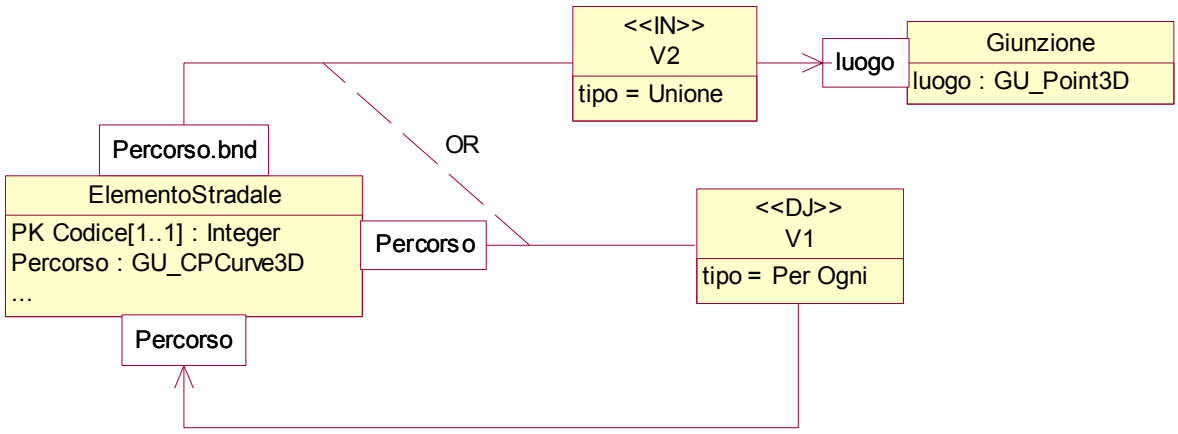
Dal punto di vista della rappresentazione testuale introduciamo la seguente aggiunta alla sintassi per rappresentare una disgiunzione di vincoli.

Dal punto di vista grafico una disgiunzione di vincoli topologici esistenziali si rappresenta congiungendo con una linea i vincoli che sono in disgiunzione e indicando il simbolo $\vee$ inscritto in un cerchio ad una estermità della linea.

Rappresentazione testuale di una disgiunzione di vincoli topologici esistenziali

<vincolo topologico> → *Vtopo* <classe vincolata> <selezioneX> <boundary>
<lista relazioni> <tipo vincolo>
<classe vincolante> <selezioneYX> <boundary> <or lista vincoli>
<or lista vincoli> → *or* <lista vincoli dis> | ε
<lista vincoli dis> → <vincolo topologico>
<lista vincoli dis> → <vincolo topologico> *or* <lista vincoli dis>

Rappresentazione grafica di una disgiunzione di vincoli topologici esistenziali (esempio)



3.4.9 Vincoli topologici sulle classi GU_CPSurfaceB3D e GU_CXSurfaceB3D

La specifica di vincoli topologici può riguardare anche attributi geometrici il cui dominio sia una classe per la rappresentazione di superfici 2D con frontiera in 3D: vale a dire, o la classe GU_CPSurfaceB3D o la classe GU_CXSurfaceB3D.

In questo caso è necessario precisare quale delle due rappresentazioni è coinvolta nel vincolo che si sta specificando e quindi l'espressione del vincolo richiede di esplicitare tale informazione. Le due rappresentazioni sono indicate con le lettere S e C, dove S rappresenta la superficie 2D e C la curva 3D, se il vincolo si esprime sull'attributo geometrico "g: GU_CPSurfaceB3D", il vincolo si esprime sulla superficie indicando come attributo geometrico g.S oppure sulla curva indicando g.C.

La sintassi della rappresentazione testuale di un vincolo topologico deve essere di conseguenza così estesa:

<classe vincolata> → <identificatore>.<identificatore_attr_geo> |
<identificatore>.<identificatore_attr_geo>.*S* |
<identificatore>.<identificatore_attr_geo>.*B*
<classe vincolante> → <identificatore>.<identificatore_attr_geo> |
<identificatore>.<identificatore_attr_geo>.*S* |
<identificatore>.<identificatore_attr_geo>.*B* |
<identificatore>.<id ruolo>.<identificatore_attr_geo> |
<identificatore>.<id ruolo>.<identificatore_attr_geo>.*S* |
<identificatore>.<id ruolo>.<identificatore_attr_geo>.*B*

<identificatore_attr_geo> → <identificatore>
<id ruolo> → <identificatore> | <identificatore>*::*<identificatore>

3.5 Rappresentazione in OCL delle relazioni topologiche

Il vincolo di integrità topologico, sia nella versione esistenziale che in quella universale, richiede in tutte le sue diverse forme la verifica dell'esistenza, tra gli oggetti delle classi coinvolte nel vincolo, di una relazione topologica dell'insieme REL_{topo} (vedi Definizione 41). In particolare, in tutte le definizioni di vincolo specificate nella sezione precedente si è supposta presente una funzione *check(a, {rel$_1$, ...,rel$_k$}, b)* che sia in grado di verificare la presenza di una certa disgiunzione di relazioni topologiche di REL_{topo} tra due oggetti di appartenenti alla classe GU_Object.

In generale l'espressione booleana che corrisponde alla funzione *check(a, {rel$_1$, ...,rel$_k$}, b)* può essere rappresentata in OCL nel seguente modo:

(rel$_1$(a,b) or ... or rel$_k$(a,b))

dove *rel$_i$(a,b)* ,$1 \le i \le k$, è l'espressione in OCL della relazione topologica *rel$_i$* dell'insieme *REL$_{topo}$*. La definizione formale in OCL di *rel$_i$(a,b)* dipende, per alcuni tipi di relazioni, dalla classe di appartenenza degli oggetti *a* e *b*. Nei successivi paragrafi si presentano tali definizioni nei diversi casi che si possono presentare.

3.5.1 La rappresentazione delle relazioni topologiche Disjoint, In, Contains e Equal in OCL

In questa sezione si specificano le espressioni booleane OCL che corrispondono alla funzione *rel$_i$(a,b)* per il caso in cui *rel$_i$* sia: *disjoint (DJ), in (IN), containts (CT)* e *equal (EQ)* ed *a* e *b* siano oggetti appartenenti a una sottoclasse di GU_Object.

Tale definizione può essere espressa usando solo funzioni della classe GM_Object da cui tutte le altre classi ereditano. In particolare, nella definizione viene usata la funzione *bRelate* definita nello standard ISO 19107 (clause 8.2).

Definizione 48. Relazioni topologiche DJ, IN, CT e EQ in OCL

Le relazioni DJ, IN, CT e EQ tra due oggetti a e b possono essere espresse in OCL attraverso la funzione bRelate definita nello standard ISO 19107 nel seguente modo:

DJ(a,b) ≡ bRelate(a, b, "FTTT")

IN(a,b) ≡ bRelate(a, b, "TFTT")

CT(a,b) ≡ bRelate(a, b, "TTFT")

EQ(a,b) ≡ bRelate(a, b, "TFFT")

La stringa "XXXX" che compare come terzo parametro della funzione bRelate rappresenta la configurazione delle matrice che definisce la relazione topologica tra a e b di cui si vuole verificare la presenza. La sequenza di quattro caratteri va interpretata come la concatenazione delle righe di una matrice 2 x 2, i cui elementi rappresentano il risultato dell'intersezione tra la chiusura (closure) e il complemento (exterior) dei due oggetti a e b.

$$\begin{pmatrix} a.closure() \cap b.closure() & a.closure() \cap b.exterior() \\ a.exterior() \cap b.closure() & a.exterior() \cap b.exterior() \end{pmatrix}$$

I valori possibili per valutare il risultato dell'intersezione sono:

- T: intersezione diversa dall'insieme vuoto
- F: intersezione uguale all'insieme vuoto
- N: qualsiasi

Ad esempio la stringa "FTTT" indica che l'intersezione tra le chiusure dei due oggetti è vuota mentre tutte le altre intersezioni sono non vuote. E' ovvio che per oggetti di dimensioni finite l'intersezione tra i complementi è sempre non vuota. Quindi tutte le matrici hanno "T" nell'ultimo elemento.

Per una descrizione della semantica della funzione bRelate si veda la clausola 8.2 dello standard ISO 19107.

3.5.2 La rappresentazione delle relazioni topologiche Touch, Overlap e Cross in OCL per oggetti di tipi geometrici dimensionalmente omogenei

In questa sezione si specificano le espressioni booleane OCL che corrispondono alla funzione $rel_i(a,b)$ per il caso in cui rel_i sia: *Touch(TC), Overlap(OV)* e *Cross(CR)* ed *a* e *b* siano oggetti appartenenti a tipi geometrici di GeoUML dimensionalmente omogenei che ereditano dalla classe GM_Complex.

Tali relazioni topologiche non possono essere definite per tutte le classi dello Spatial Schema. Ciò è dovuto al fatto che la funzione *cRelate()* contenuta nello standard ISO 19107, che consente di verificare la presenza di una data relazione topologica tra due oggetti (secondo le definizioni della letteratura di Egenhofer e Clementini), non è applicabile ad oggetti di dimensione non omogenea (oggetti delle classi GM_Aggregate e GM_Complex) come discusso nel paragrafo 2.5.1.

La definizione in OCL si basa sulla funzione *cRelate()* prevista dallo standard ISO 19107 (clause 8.4).

Definizione 49. Relazioni topologiche TC, OV e CR per i tipi geometrici dimensionalmente omogenei che ereditano dalla classe GM_Complex [Clementini93].

Le relazioni TC, OV e CR tra due oggetti a e b appartenenti a tipi dimensionalmente omogenei che ereditano dalla classe GM_Complex possono essere espresse in OCL attraverso la funzione cRelate dello standard ISO 19107 nel seguente modo:

TC(a,b) ≡ cRelate(a, b, "0NN-NFN-NN2") or cRelate(a, b, "1NN-NFN-NN2") or cRelate(a, b, "F0N-NFN-NN2") or cRelate(a, b, "F1N-NFN-NN2") or cRelate(a, b, "FNN-0FN-NN2") or cRelate(a, b, "FNN-1FN-NN2")

CR(a,b) ≡ eRelate(a, b, "NN1-N12-N12") or cRelate(a, b, "NNN-N01-N12")

OV(a,b) ≡ CR(a,b) or cRelate(a, b, "011-122-122") or cRelate(a, b, "111-122-122") or cRelate(a, b, "NNN-N11-N12")

La stringa "XXX-XXX-XXX" che compare come terzo parametro della funzione cRelate rappresenta la configurazione delle matrice che definisce la relazione topologica tra a e b di cui si vuole verificare la presenza. La sequenza di nove caratteri va interpretata come la concatenazione delle righe di una matrice 3 x 3, i cui elementi rappresentano il risultato dell'intersezione tra il boundary, la parte interna (interior) e il complemento (exterior) dei due oggetti a e b.

$$\begin{pmatrix} a.boundary() \cap b.boundary() & a.boundary() \cap b.\text{int}erior() & a.boundary \cap b.exterior() \\ a.\text{int}erior() \cap b.boundary() & a.\text{int}erior() \cap b.\text{int}erior() & a.\text{int}erior() \cap b.exterior() \\ a.exterior() \cap b.boundary() & a.exterior() \cap b.\text{int}erior() & a.exterior() \cap b.exterior() \end{pmatrix}$$

I valori possibili per valutare il risultato dell'intersezione sono:

- 0, 1, 2, 3: intersezione diversa dall'insieme vuoto e costituita rispettivamente solo da punti, linee, poligoni o solidi.
- F: intersezione uguale all'insieme vuoto
- N: qualsiasi

Per una descrizione della semantica della funzione bRelate si veda la clausola 8.2 dello standard ISO 19107

Tale definizione risulta quindi applicabile agli oggetti delle seguenti classi: GU_CPCurve2D, GU_CPCurve3D, GU_CPSurface2D, GU_CXPoint2D, GU_CXPoint3D, GU_CXCurve2D, GU_CXCurve3D, GU_CXSurface2D.

3.5.3 La rappresentazione in OCL della relazione topologica Intersects per oggetti di tipi geometrici non dimensionalmente omogenei.

Se consideriamo generici oggetti della classe GM_Complex o GM_Aggregate, la definizione sopra riportata per le relazioni Touch (TC), Overlap (OV) e Cross (CR) non è corretta. Infatti, la funzione *cRelate* non può essere applicata a oggetti di tali classi (ISO 19107 - clause 8.1), in quanto tali oggetti possono essere dimensionalmente non omogenei.

Può invece essere applicata a tali oggetti la relazione Intersects, la cui definizione in OCL è la seguente:

Definizione 50. Relazione topologica Intersects per oggetti di tipi geometrici dimensionalmente non omogenei che ereditano dalla classe GM_Aggregate e GM_Complex

La relazione Intersects (INT), corrispondente alla disgiunzione (TC or OV), tra due oggetti a e b appartenenti alla classe GM_Aggregate o alla classe GM_Complex può essere espressa in OCL, attraverso le funzioni intersects e contains definite sulla classe GM_Object dello standard ISO 19107, nel seguente modo:

INT(a,b) ≡ a.closure().intersects(b.closure()) and
not a.closure().contains(b.closure()) and not b.closure().contains(a.closure())

3.6 Vincoli di struttura sugli oggetti complessi e aggregati

Per garantire la consistenza nella rappresentazione degli oggetti delle classi che ereditano da GM_Complex e GM_Aggregate è necessario introdurre alcuni vincoli di integrità aggiuntivi che caratterizzano il legame tra la classe composta e la classe componente.

Tali vincoli di integrità sono vincoli di tipo esistenziale: essi richiedono che dato l'attributo geometrico *g* di un oggetto della classe composta esistano nella classe componente gli oggetti con attributo geometrico *f* componenti *g* o viceversa.

In particolare, i vincoli di base sono di due tipi:

- il vincolo di consistenza che esprime il fatto che i componenti devono appartenere al composto: *Appartiene*
- il vincolo di consistenza che esprime il fatto che il composto deve essere generato da un certo numero di componenti: *CompostoDa*

In Figura 10 sono riportate le classi ed associazioni dello Spatial Schema che sono rilevanti per la comprensione delle seguenti definizioni di vincolo tra due ipotetiche classi X e Y dotate di attributi geometrici g ed f di un tipo geometrico di GeoUML che eredita da GM_Complex. In figura si è riportata direttamente la classe padre GM_Complex.

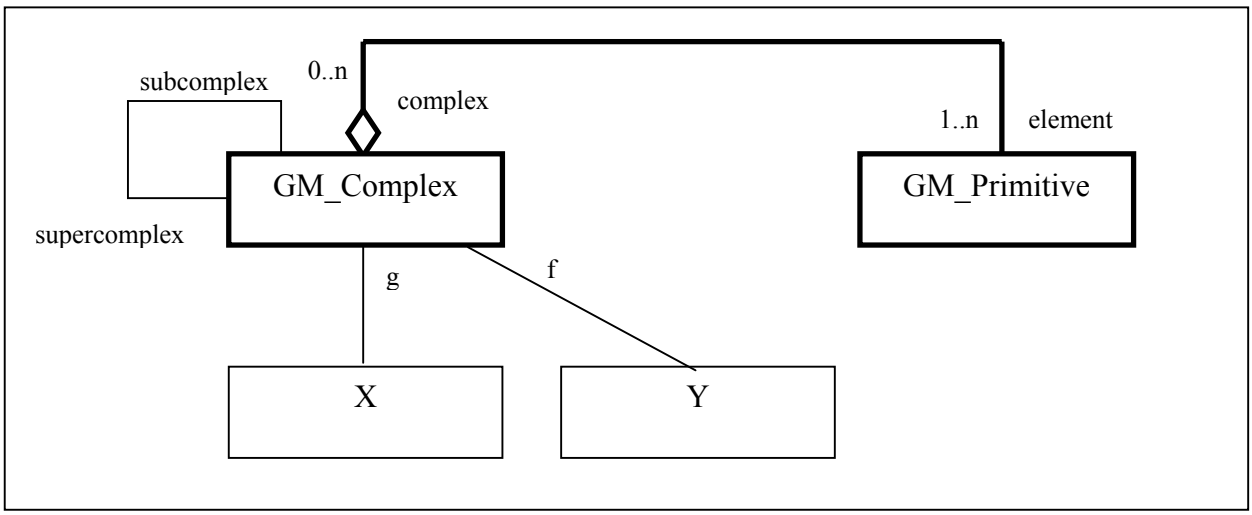


Figura 10 Rappresentazione di due classi X e Y con attributi di tipo GM_Complex.

3.6.1 Il vincolo di struttura Appartiene

Il vincolo di struttura *Appartiene(X, g, Y, f)* definisce un vincolo di appartenenza dell'attributo geometrico *g* di ogni istanza della classe *X* all'attributo geometrico *f* di almeno un'istanza di *Y*.

La definizione formale cambia a seconda della classe di GeoUML a cui appartengono gli attributi *g* e *f*. Infatti tale vincolo di appartenenza ha significati diversi nei diversi casi, in particolare:

- se *g* e *f* sono entrambi oggetti di classi che ereditano da GM_Complex, la relazione di appartenenza corrisponde ad una relazione sottocomplesso-supercomplesso;
- infine se *g* è di una classe che eredita da GM_Object e *f* è di una classe che eredita da GM_Aggregate, allora la composizione si genera a partire dalla relazione elemento-aggregato.

Definizione 51. Vincolo Appartiene tra attributi di un tipo geometrico che eredita da GM_Complex

Data una classe X con attributo geometrico g di tipo GM_Complex e una classe Y con attributo geometrico f di tipo GM_Complex, il vincolo Appartiene(X, g, Y, f) è definito dalla seguente espressione in OCL:

Appartiene(X, g, Y, f) ≡

context X

inv: *Y.allinstances→exists(a: Y | self.g.supercomplex→includes(a.f))*

In Figura 10 sono rappresentate le classi dello Spatial Schema coinvolte nel vincolo.

Rappresentazione testuale del vincolo (le regole fuori dal riquadro sono già state presentate in precedenza)

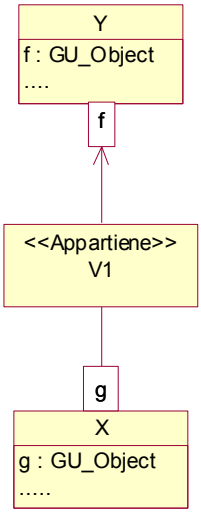
* <vincolo di struttura> → *struttura* <classe vincolata> *appartiene* <classe vincolante>

<classe vincolata> → <identificatore>.<identificatore_attr_geo> |
<identificatore>.<identificatore_attr_geo>.*S* |
<identificatore>.<identificatore_attr_geo>.*B*

* <classe vincolante> → <identificatore>.<identificatore_attr_geo> |
<identificatore>.<identificatore_attr_geo>.*S* |
<identificatore>.<identificatore_attr_geo>.*C*

<identificatore_attr_geo> → <identificatore>

Rappresentazione grafica del vincolo



Nel caso in cui invece gli attributi coinvolti nel vincolo siano uno di tipo GM_Object e l'altro di tipo GM_Aggregate il vincolo è definito nel seguente modo.

Definizione 52. Vincolo Appartiene tra un attributo di un tipo che eredita da GU_Object e un attributo di un tipo geometrico che eredita da GM_Aggregate.

Data una classe X con attributo geometrico g di tipo GM_Object e una classe Y con attributo geometrico f di tipo GM_Aggregate, il vincolo Appartiene(X, g, Y, f) è definito dalla seguente espressione in OCL:

Appartiene(X, g, Y, f) ≡

context X

inv: *Y.allinstances→ exists(a: Y | a.f.element→ includes(self.g))*

In Figura 11 sono rappresentate le classi dello Spatial Schema coinvolte nel vincolo. La rappresentazione grafica del vincolo è uguale a quella del caso precedente.

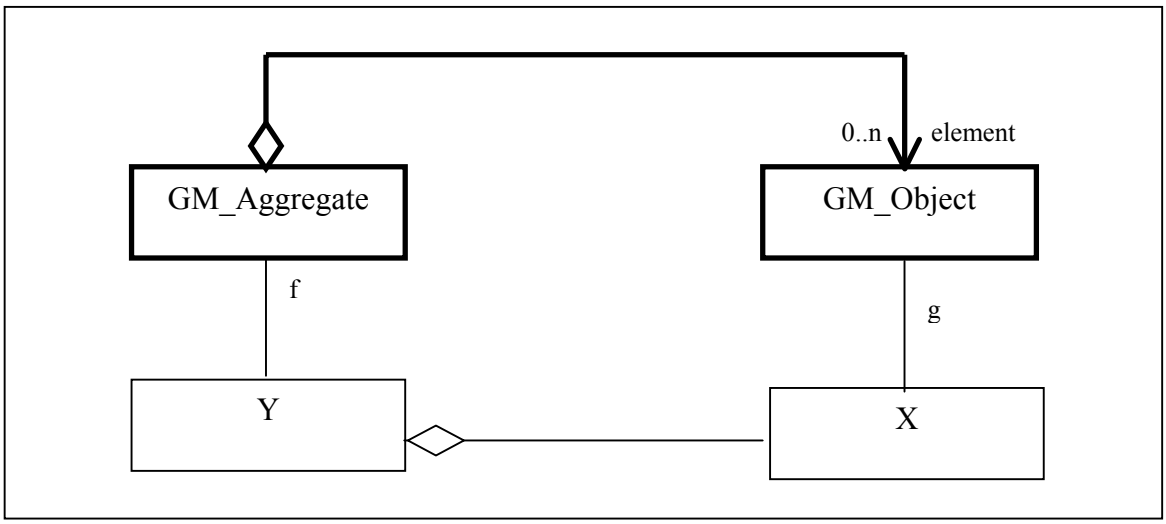


Figura 11 Rappresentazione di due classi X e Y con attributi geometrici GM_complex e GM_Primitive rispettivamente

In molti casi, oltre a richiedere che gli oggetti geometrici di una classe siano sottocomplessi degli oggetti geometrici di un'altra classe, si richiede anche che gli oggetti geometrici che sono sottocomplessi dello stesso oggetto abbiano insiemi di primitive interne (cioè primitive che non appartengono al boundary) disgiunti. Ovviamente tale variante del vincolo di appartenenza può essere applicato solo se *g* e *f* sono entrambi oggetti di classi che ereditano da GM_Complex e g è dimensionalmente omogeneo.

Tale situazione viene espressa in GeoUML introducendo il vincolo di appartenenza disgiunta definito come segue.

Definizione 53. Vincolo DJ-Appartiene tra attributi di tipo GM_Complex

Data una classe X con attributo geometrico g di tipo GM_Complex e una classe Y con attributo geometrico f di tipo GM_Complex, il vincolo DJ-Appartiene(X, g, Y, f) è definito dalla seguente espressione in OCL:

DJ-Appartiene(X, g, Y, f) ≡

context X

inv: *Y.allinstances→exists(a: Y | self.g.supercomplex→includes(a.f)) and*
X.allinstances→forall(x: X | brother(x.g, self.g, Y, f) implies DJ-str(x.g, self.g))

dove:

brother(x, y, Z, c) ≡Z.allinstances→exists(z: Z | x.supercomplex→includes(z.c) and
y.supercomplex→includes(z.c))

DJ-str(x, y) ≡x.element→forall(z: GM_Primitive |
not (y.element→includes(z)) or
x.boundary().element→includes(z) or
y.boundary().element→includes(z))

Una proprietà più debole della completa disgiunzione delle primitive interne è la seguente, che chiameremo "quasi-disgiunzione": due oggetti complessi della stessa dimensione sono quasi-disgiunti se i sottoinsiemi di primitive di dimensione massima sono disgiunti.

Tale situazione viene espressa in GeoUML introducendo il vincolo di appartenenza quasi disgiunta definito come segue. Tale definizione è molto simile alla precedente definizione di vincolo DJ-Appartiene: la sola differenza sta nel predicato QDJ-str() che sostituisce il predicato DJ-str().

Definizione 54. Vincolo QDJ-Appartiene tra attributi di un tipo geometrico che eredita da GM_Complex

Data una classe X con attributo geometrico g di tipo GM_Complex e una classe Y con attributo geometrico f di tipo GM_Complex, il vincolo QDJ-Appartiene(X, g, Y, f) è definito dalla seguente espressione in OCL:

QDJ-Appartiene(X, g, Y, f) ≡

context X

inv: *Y.allinstances→exists(a: Y | self.g.supercomplex→includes(a.f)) and*
X.allinstances→forall(x: X | brother(x.g, self.g, Y, f) implies QDJ-str(x.g, self.g))

dove:

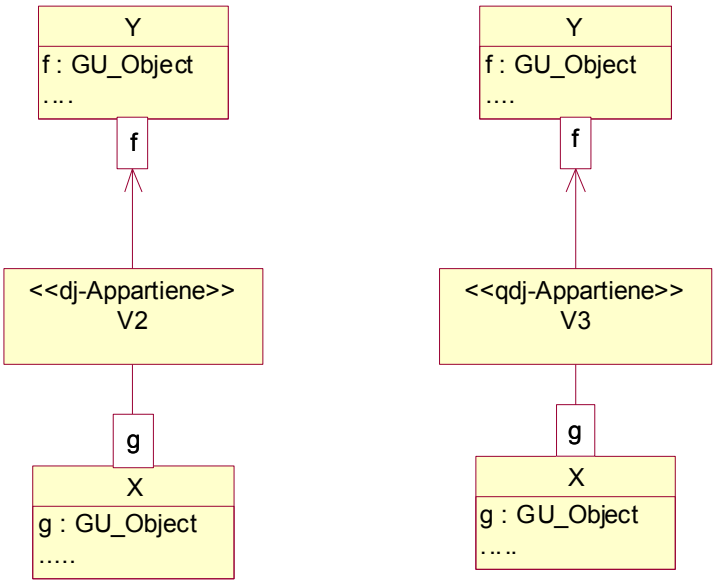
brother(x, y, Z, c) è uguale al predicato della definizione precedente,

QDJ-str(x, y) ≡ x.element→forall(z: GM_Primitive |
not (y.element→includes(z)) or
x.boundary().element→includes(z) or
y.boundary().element→includes(z) or
z.oclAsType(GM_Point))

Rappresentazione testuale del vincolo nei casi dj-appartiene e qdj-appartiene

* <vincolo di struttura> → *struttura* <classe vincolata> *dj-appartiene* <classe vincolante>
* <vincolo di struttura> → *struttura* <classe vincolata> *qdj-appartiene* <classe vincolante>

Rappresentazione grafica del vincolo nei casi dj-appartiene e qdj-appartiene



Infine nel caso in cui il vincolo DJ-Appartiene coinvolga una attributo di tipo GU_CPSurfaceB3D o GU_CXSurfaceB3D è possibile indicare che il vincolo abbia un effetto sia sulla rappresentazione della superficie nello spazio 2D sia sulla corrispondente curva nello spazio 3D. Precisamente si può imporre che se due superfici sono adiacenti nello spazio 2D, le curve 3D corrispondenti condividano la primitiva che corrisponde, proiettata nello spazio 2D, alla porzione di boundary condivisa dalle due superfici. Tale vincolo viene formalmente precisato nel seguente modo.

Definizione 55. Vincolo DJ-Appartiene tra attributi di tipo GU_CXSurfaceB3D

Data una classe X con attributo geometrico g di tipo GU_CXSurfaceB3D e una classe Y con attributo geometrico f di tipo GU_CXSurfaceB3D, il vincolo DJ-Appartiene(X, g, Y, f) è definito dalla seguente espressione in OCL:

DJ-Appartiene(X, g, Y, f) ≡

context X

inv: *Y.allinstances→exists(a: Y | self.g.S.supercomplex→includes(a.f.S)) and*
X.allinstances→forall(x: X | brotherB3D(x.g, self.g, Y, f) implies DJ-strB3D(x.g, self.g))

dove:

brotherB3D(x, y, Z, c) ≡ Z.allinstances→ exists(z: Z | x.S.supercomplex→includes(z.c.S) and
y.S.supercomplex→includes(z.c.S))

DJ-str(x, y) ≡ x.S.element→ forall(z: GM_Primitive |
not (y.S.element→includes(z)) or
(x.S.boundary().element→includes(z) and
y.S.boundary().element→includes(z) and
x.B.element→ exists(c_1: GM_Primitive| c_1.planar().equals(z)) and
y.B.element→ exists(c_2: GM_Primitive| c_2.planar().equals(z))
))

Se uno dei due attributi non è di tipo GU_CPSurfaceB3D o GU_CXSurfaceB3D la definizione cambia solo per il fatto che la componente S non viene più precisata nella definizione del vincolo per l'attributo *g* o *f* che non è di tipo GU_CPSurfaceB3D o GU_CXSurfaceB3D.

Per applicare questa variante del vincolo DJ-Appartiene nella specifica testuale del vincolo non si deve precisare la componente di riferimento (S) sull'attributo di tipo GU_CPSurfaceB3D o GU_CXSurfaceB3D, Ad esempio:

classe X *attributi:* g: GU_CXSurfaceB3D.

classe Y *attributi:* f: GU_CXSurface2D.

struttura X.g.S *dj-appartiene* Y.f (vincolo normale solo sulla componente S)

struttura X.g *dj-appartiene* Y.f (variante del vincolo con effetto anche sulla componente B nello spazio 3D)

3.6.2 Il vincolo di struttura Appartiene collegato ad una associazione

Il vincolo di struttura *AppartieneSuAssociazione(X, g, r, Y, f)* posto sulla classe *X* definisce un vincolo di inclusione dell'attributo geometrico *g* di ogni istanza di *X (x)* nell'attributo geometrico *f* di almeno un'istanza di *Y (y)* collegata a *x* attraverso il ruolo *r*.

Come per il vincolo di struttura *Appartiene* non collegato ad una associazione presentato precedentemente esistono diverse versioni della definizione formale.

Definizione 56. Vincolo Appartiene tra attributi di un tipo geometrico che eredita da GM_Complex con associazione

Data una classe X con attributo geometrico g di tipo GM_Complex e un ruolo r di tipo Y, e una classe Y con attributo geometrico f di tipo GM_Complex, il vincolo AppartieneSuAssociazione(X, g, r, Y, f) è definito dalla seguente espressione in OCL:

AppartieneSuAssociazione(X, g, r, Y, f) ≡

context X

inv: *self.r → exists(a: Y | self.g.supercomplex→includes(a.f))*

Definizione 57. Vincolo Appartiene tra un attributo di un tipo geometrico che eredita da GU_Object e un attributo di un tipo geometrico che eredita da GM_Aggregate con associazione

Data una classe X con attributo geometrico g di tipo GM_Object e un ruolo r di tipo Y, e una classe Y con attributo geometrico f di tipo GM_Aggregate, il vincolo AppartieneSuAssociazione(X, g, r, Y, f) è definito dalla seguente espressione in OCL:

Appartiene(X, g, r, Y, f) ≡

context X

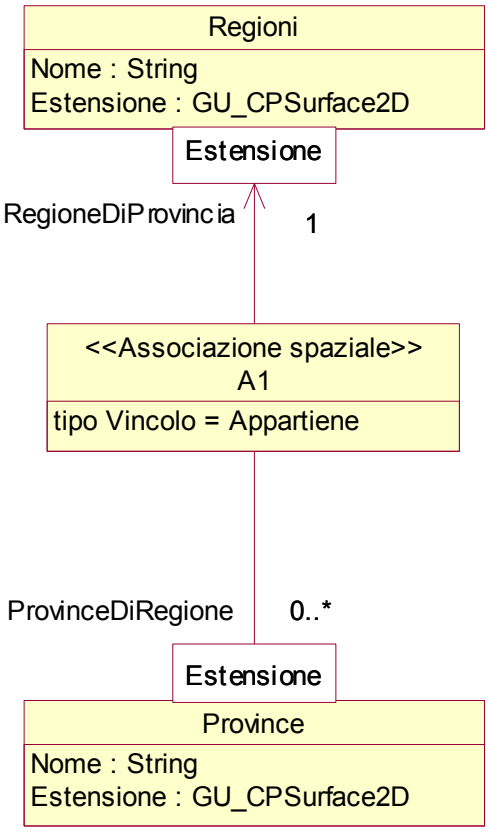
inv: *self.r* → *exists(a: Y | a.f.element*→ *includes(self.g))*

Si noti che l'unica differenza rispetto alle definizioni della sezione precedente consiste nella individuazione degli oggetti a cui applicare la condizione *exist(...)*. Infatti, mentre nelle definizioni della sezione precedente gli oggetti erano individuati con la clausola *Y.allinstances*, che rappresenta tutte le istanze della classe *Y*, nelle definizioni di questa sezione viene utilizzata la clausola *self.r*, che rappresenta invece gli oggetti della classe *Y* legate all'oggetto corrente *self* attraverso il ruolo *r*.

Rappresentazione testuale del vincolo (tutte le regole sono già state presentate in precedenza)

***** <vincolo di struttura> → *struttura* <classe vincolata> *appartiene* <classe vincolante>
<classe vincolata> → <identificatore>.<identificatore_attr_geo> |
<identificatore>.<identificatore_attr_geo>.*S* |
<identificatore>.<identificatore_attr_geo>.*B*
<classe vincolante> → <identificatore>.<identificatore_attr_geo> |
<identificatore>.<identificatore_attr_geo>.*S* |
<identificatore>.<identificatore_attr_geo>.*B* |
<identificatore>.<id ruolo>.<identificatore_attr_geo> |
<identificatore>.<id ruolo>.<identificatore_attr_geo>.*S* |
<identificatore>.<id ruolo>.<identificatore_attr_geo>.*B*
<identificatore_attr_geo> → <identificatore>
<id ruolo> → <identificatore> | <identificatore>::<identificatore>

Rappresentazione grafica del vincolo (esempio)



3.6.3 Il vincolo di struttura CompostoDa

Il vincolo di struttura *CompostoDa(Y, f, X, g)* definisce un vincolo di composizione dell'attributo geometrico *f* di ogni istanza di *Y*. Tale vincolo stabilisce che *f* si ottenga dall'unione degli attributi geometrici *g* di una o più istanze di *X*.

La definizione formale cambia a seconda della classe dello Spatial Schema a cui appartengono gli attributi *f* e *g*. Infatti la composizione si ottiene in modo diverso nei due casi possibili che si possono verificare in GeoUML:

- se *f* e *g* sono entrambi oggetti della classe GM_Complex, la composizione si ottiene dalla relazione sottocomplesso-supercomplesso ("*g subcomplex of f*");
- se *g* è della classe GM_Object e *f* è della classe GM_Aggregate, allora la composizione si genera a partire dalla relazione elemento-aggregato.

Definizione 58. Vincolo CompostoDa tra attributi di un tipo geometrico che eredita da GM_Complex

Data una classe Y con attributo geometrico f di tipo GM_Complex, e una classe X con attributo geometrico g di tipo GM_Complex, il vincolo CompostoDa(Y, f, X, g) è definito dalla seguente espressione in OCL:

CompostoDa(Y, f, X, g) ≡
context Y
inv: *self.f.element→forall(e: GM_Primitive | X.allinstances→ exists(a: X | a.g.element→ includes(e)))*

Si noti che la definizione formale verifica che tutti gli oggetti di tipo GM_Primitive che costituiscono gli elementi del complesso *f* (*f.element*) siano elementi del complesso che rappresenta l'attributo geometrico *g* di almeno un'istanza di *X* (si veda Figura 10).

Definizione 59. Vincolo CompostoDa tra un attributo di un tipo geometrico che eredita da GM_Aggregate e un attributo di tipo geometrico che eredita da GU_Object

Data una classe Y con attributo geometrico f di tipo GM_Aggregate, e una classe X con attributo geometrico g di tipo GM_Object, il vincolo CompostoDa(Y, f, X, g) è definito dalla seguente espressione in OCL:

CompostoDa(Y, f, X, g) ≡
context Y
inv: *self.f.element→forall(e: GM_Object | X.allinstances→ exists(a: X | a.g = e))*

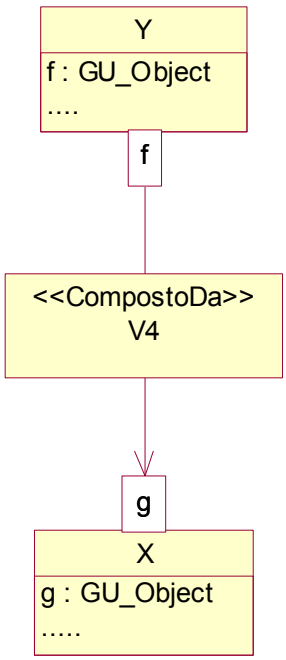
Rappresentazione testuale del vincolo (le regole fuori dal riquadro sono già state presentate in precedenza)

* <vincolo di struttura> → <u>struttura</u> <classe vincolata> <u>compostoDa</u> <classe vincolante>

<classe vincolata> → <identificatore>.<identificatore_attr_geo> |
<identificatore>.<identificatore_attr_geo>.<u>S</u> |
<identificatore>.<identificatore_attr_geo>.<u>B</u>
* <classe vincolante> → <identificatore>.<identificatore_attr_geo> |
<identificatore>.<identificatore_attr_geo>.<u>S</u> |

<identificatore>.<identificatore_attr_geo>.*B*
<identificatore_attr_geo> → <identificatore>

Rappresentazione grafica del vincolo



3.6.4 Il vincolo di struttura CompostoDa collegato ad una associazione

Il vincolo di struttura *CompostoDaSuAssociazione(Y, f, r, X, g)* definisce un vincolo di composizione dell'attributo geometrico *f* delle istanze di *Y*. Tale vincolo stabilisce che data una istanza di *Y (y)* l'attributo geometrico *f* si ottenga dall'unione degli attributi geometrici *g* di tutte le istanze di *X* collegate all'istanza di *Y* attraverso il ruolo *r*.

Come per il vincolo di struttura *CompostoDa* non collegato ad una associazione presentato nella sezione precedente esistono diverse versioni della definizione formale.

Definizione 60. Vincolo CompostoDa tra attributi di un tipo geometrico che eredita da GM_Complex con associazione

Data una classe Y con attributo geometrico f di tipo GM_Complex e un ruolo r di tipo X, e una classe X con attributo geometrico g di tipo GM_Complex, il vincolo CompostoDaSuAssociazione(Y, f, r, X, g) è definito dalla seguente espressione in OCL:

CompostoDaSuAssociazione(Y, f, r, X, g) ≡
context X
inv: *self.f.element→forall(e: GM_Primitive | self.r→*
exists(a: X | a.g.element→ includes(e)))

Si noti che la definizione in questo caso richiede che le primitive che costituiscono il complesso *f* (*f.element*) di un'istanza *y* di Y siano contenute nei complessi che rappresentano attributi *g* delle istanze di X raggiunte da *y* attraverso il ruolo *r*.

Definizione 61. Vincolo CompostoDa tra un attributo di un tipo che eredita da GM_Aggregate e un attributo di un tipo geometrico che eredita da GU_Object con associazione

Data una classe Y con attributo geometrico f di tipo GM_Aggregate e un ruolo r di tipo X, e una classe X con attributo geometrico g di tipo GM_Object, il vincolo CompostoDaSuAssociazione(Y, f, r, X, g) è definito dalla seguente espressione in OCL:

CompostoDaSuAssociazione(Y, f, r, X, g) ≡
context X
inv: *self.f.element→forall(e: GM_Object | self.r → exists(a: X | a.g = e))*

Rappresentazione testuale del vincolo (tutte le regole sono già state presentate in precedenza)

* <vincolo di struttura> → <u>*struttura*</u> <classe vincolata> <u>*compostoDa*</u> <classe vincolante>
<classe vincolata> → <identificatore>.<identificatore_attr_geo> |
<identificatore>.<identificatore_attr_geo>.<u>*S*</u> |
<identificatore>.<identificatore_attr_geo>.<u>*B*</u>
<classe vincolante> → <identificatore>.<identificatore_attr_geo> |
<identificatore>.<identificatore_attr_geo>.<u>*S*</u> |
<identificatore>.<identificatore_attr_geo>.<u>*B*</u> |
<identificatore>.<id ruolo>.<identificatore_attr_geo> |
<identificatore>.<id ruolo>.<identificatore_attr_geo>.<u>*S*</u> |
<identificatore>.<id ruolo>.<identificatore_attr_geo>.<u>*B*</u>
<identificatore_attr_geo> → <identificatore>
<id ruolo> → <identificatore> | <identificatore>::<identificatore>

Rappresentazione grafica del vincolo

Vedi vincolo appartiene con associazione

3.6.5 I vincoli Appartiene e CompostoDa con riferimento al boundary di oggetti geometrici

Tutti i vincoli di tipo Appartiene e CompostoDa possono essere applicati alla frontiera (boundary) degli attributi g ed f invece che agli attributi stessi. Naturalmente, questa possibilità è ammessa solamente per i tipi che hanno boundary definito (si veda il paragrafo 2.5.2): si escludono quindi gli oggetti delle classi che ereditano da GM_Aggregate e gli oggetti delle classi GU_Complex2D e GU_Complex3D.
Per considerare il boundary degli oggetti, le definizioni date nei precedenti paragrafi 3.6.1, 3.6.2, 3.6.3 e 3.6.4 vanno modificate introducendo la funzione boundary e ricordando che tale funzione restituisce un oggetto della classe GM_Complex. Si riportano di seguito le definizioni modificate.

Definizione 62. Vincolo Appartiene tra il boundary di un attributo di tipo GU_CXSurface2D (o GU_CPSurface2D o GU_CXCurve* o GU_CPCurve*) e un attributo di un tipo geometrico che eredita da GM_Complex

Data una classe X con attributo geometrico g di tipo GU_CXSurface2D (o GU_CXCurve o GU_CP*) e una classe Y con attributo geometrico f di tipo GM_Complex, il vincolo* $Appartiene^{B-}(X, g, Y, f)$ *è definito dalla seguente espressione in OCL:*

$Appartiene(X, g, Y, f)^{B-} \equiv$

context X

inv: *Y.allinstances→exists(a: Y | self.g.boundary().supercomplex→includes(a.f))*

Definizione 63. Vincolo Appartiene tra un attributo di un tipo geometrico che eredita da GM_Complex e il boundary di un attributo di tipo GU_CXSurface2D (o GU_CPSurface2D o GU_CXCurve* o GU_CPCurve*)

Data una classe X con attributo geometrico g di tipo GM_Complex e una classe Y con attributo geometrico f di tipo GU_CXSurface2D (o GU_CXCurve o GP_CPCurve* o GU_CPSurface2D), il vincolo* $Appartiene^{-B}(X, g, Y, f)$ *è definito dalla seguente espressione in OCL:*

$Appartiene(X, g, Y, f)^{-B} \equiv$

context X

inv: *Y.allinstances→exists(a: Y | self.g.supercomplex→includes(a.f.boundary()))*

Anche le definizioni dei vincoli CompostoDa vanno modificate nel caso in cui si riferiscano al boundary degli attributi geometrici come segue.

Definizione 64. Vincolo CompostoDa tra il boundary di un attributo di tipo geometrico GU_CXSurface2D (o GU_CPSurface2D o GU_CXCurve* o GU_CPCurve*) e un attributo di tipo geometrico GU_CXCurve* (o GU_CPCurve* o GU_CXPoint*)

Data una classe Y con attributo geometrico f di tipo GU_CXSurface2D (o GU_CXCurve o GU_CPCurve* o GU_CPSurface2D), e una classe X con attributo geometrico g di tipo GU_CXCurve* (o GU_CPCurve* o GU_CXPoint*), il vincolo* $CompostoDa^{B-}(Y, f, X, g)$ *è definito dalla seguente espressione in OCL:*

$CompostoDa^{B-}(Y, f, X, g) \equiv$

context Y

inv: *self.f.boundary().element→forall(e: GM_Primitive | X.allinstances→*

exists(a: X | a.g.element→ includes(e)))

Definizione 65. Vincolo CompostoDa tra un attributo di tipo GU_CXCurve* (o GU_CPCurve* o GU_CXPoint*) e il boundary di un attributo di tipo GU_CXSurface2D (o GU_CPSurface2D o GU_CXCurve* o GU_CPCurve*)

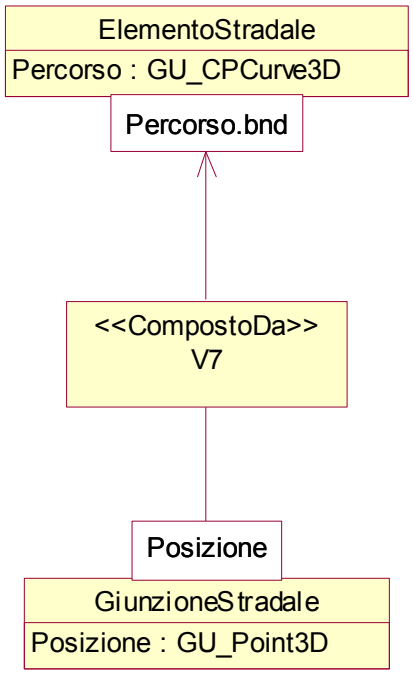
Data una classe Y con attributo geometrico f di tipo GU_CXCurve (o GU_CPCurve* o GU_CXPoint*), e una classe X con attributo geometrico g di tipo GU_CXSurface2D (o GU_CPSurface2D o GU_CXCurve* o GU_CPCurve*), il vincolo CompostoDa^{-B}(Y, f, X, g) è definito dalla seguente espressione in OCL:*

CompostoDa^{-B}(Y, f, X, g) ≡

context Y

inv: *self.f.element→forall(e: GM_Primitive | X.allinstances→*
exists(a: X | a.g.boundary().element→ includes(e)))

Graficamente viene utilizzata la notazione adottata per i vincoli topologici. Ad esempio per il vincolo CompostoDa tra il boundary dell'attributo geometrico Percorso della classe ElementoStradale e l'attributo Posizione della classe GiunzioneStradale si ha la seguente rappresentazione grafica.



La rappresentazione testuale dei vincoli Appartiene e CompostoDa è in questo caso:

Rappresentazione testuale del vincolo

* <vincolo di struttura> → *struttura* <classe vincolata> <boundary str>
<tipo vincolo str> <classe vincolante> <boundary str>
<boundary str> → ε | *.bnd*
* <tipo vincolo str> → *appartiene* | *compostoDa* | *appartiene* | *dj-appartiene* | *qdj-appartiene*

3.6.6 Il vincolo di partizione

Talvolta è necessario esprimere un vincolo indicante che l'attributo geometrico *f* di una classe *Y* viene "partizionato" in elementi che costituiscono l'attributo geometrico *g* di un'altra classe *X*; in altri termini il vincolo esprime il fatto che dato un oggetto *y* della classe *Y* esiste un certo sottoinsieme di oggetti della classe *X* i cui attributi geometrici costituiscono una partizione dell'attributo geometrico di *y*. Una partizione ha le seguenti proprietà:

- l'unione degli oggetti della classe *X* che partizionano un oggetto della classe *Y* forma l'oggetto della classe *Y*;
- gli oggetti della classe *X* che formano la partizione non si sovrappongono (al più sono adiacenti).

Ciò si esprimerebbe con i seguenti due vincoli presentati nelle precedenti sezioni:

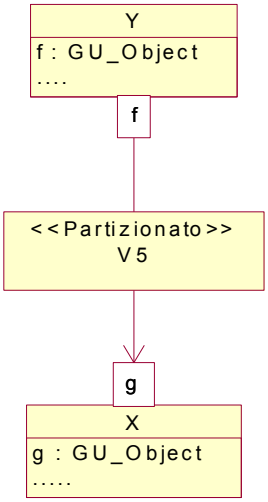
- *compostoDa (Y, f, X, g)*
- *dj-appartiene (X, g, Y, f)*

Tuttavia, data la frequenza di uso di questa coppia di vincoli, si introduce la seguente forma abbreviata:

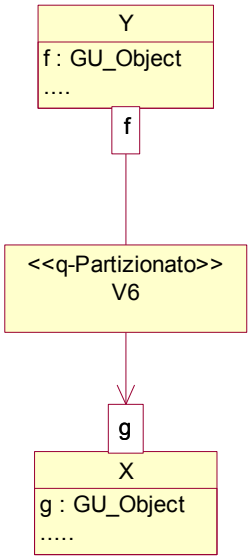
partizionato(Y, f, X, g)

che sostituisce i due vincoli sopra elencati.

Graficamente la rappresentazione risulta semplificata nel modo seguente (l'indicazione di f nella rappresentazione grafica del vincolo è necessaria per indicare quale è l'attributo che viene partizionato):



Poichè il vincolo *dj-appartiene* esiste anche nella forma meno restrittiva *qdj-appartiene*, applicabile solo a attributi di un tipo geometrico che rappresenta linee, è stata introdotta anche una versione aggiuntiva del vincolo partizione dove il vincolo *qdj-appartiene* prende il posto del vincolo *dj-appartiene*. In tal caso il vincolo partizione viene indicato con la parola chiave: *q-partizionato*.



Il vincolo di Partizione può essere applicato anche con riferimento ad una associazione quando il vincolo *compostoDa* e il vincolo *dj-appartiene* sono a loro volta collegati ad una associazione. La rappresentazione grafica in questo caso è simile a quella del vincolo Appartiene collegato ad una associazione dove il tipo vincolo diventa '*Partizionato*' oppure '*q-Partizionato*'.

Infine, il vincolo di partizione può essere applicato al boundary dell'attributo della classe partizionata oppure al boundary della classe che rappresenta gli elementi della partizione. In questo caso la definizione va modificata usando i vincoli di struttura nella versione riferita al boundary.
Si prevede una rappresentazione testuale anche per il vincolo Partizione, come segue.

Rappresentazione testuale del vincolo (le regole fuori dal riquadro sono già state presentate in precedenza)

* <vincolo di struttura> → *struttura* <classe vincolata> <boundary str>
<tipo vincolo str> <classe vincolante> <boundary str>

<tipo vincolo str> → *appartiene* | *compostoDa* | *appartiene* | *dj-appartiene* | *qdj-appartiene* | *partizionato* | *q-partizionato*

<classe vincolata> → <identificatore>.<identificatore_attr_geo> |
<identificatore>.<identificatore_attr_geo>.*S* |
<identificatore>.<identificatore_attr_geo>.*B*

<classe vincolante> → <identificatore>.<identificatore_attr_geo> |
<identificatore>.<identificatore_attr_geo>.*S* |
<identificatore>.<identificatore_attr_geo>.*B* |
<identificatore>.<id ruolo>.<identificatore_attr_geo> |
<identificatore>.<id ruolo>.<identificatore_attr_geo>.*S* |
<identificatore>.<id ruolo>.<identificatore_attr_geo>.*B*

<identificatore_attr_geo> → <identificatore>

<id ruolo> → <identificatore> | <identificatore>::<identificatore>

<boundary str> → ε | *.bnd*

3.6.7 Vincoli con riferimento all'unione di più classi

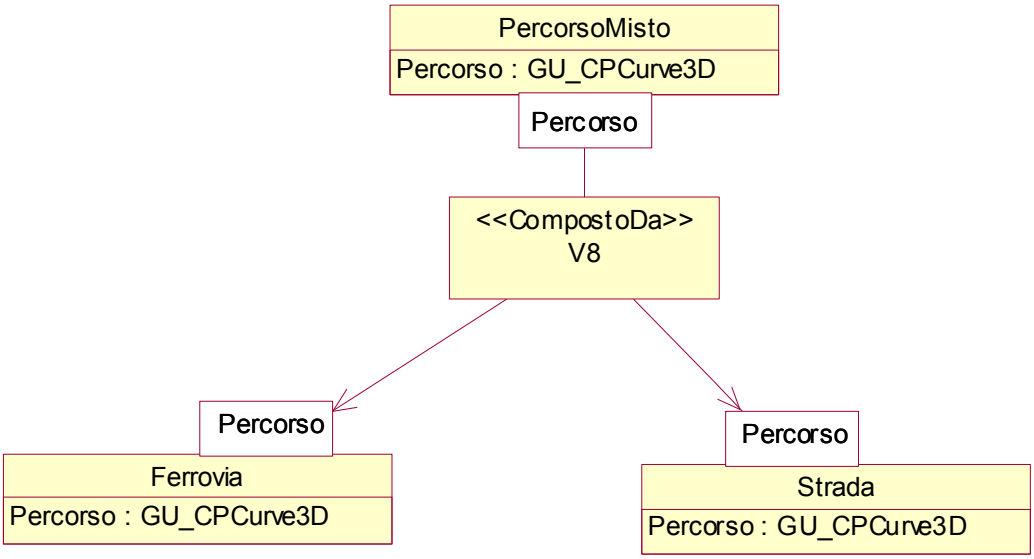
In alcuni casi si vuole che un vincolo *compostoDa* oppure *partizionato* facciano riferimento a diversi attributi geometrici di diverse classi. Si vuole indicare in questo modo la situazione nella

quale un oggetto della classe composta o partizionata è composto/partizionato in oggetti che possono appartenere a più classi diverse.

Ciò equivale ad organizzare le classi vincolanti in una gerarchia di ereditarietà introducendo una classe padre fittizia (classe stereotipata <<*abstract*>>) con il solo scopo di raggruppare le istanze delle classi figlie. Per evitare di introdurre tale gerarchia di ereditarietà si introduce la seguente variante nella sintassi testuale di un vincolo di struttura:

<vincolo di struttura> →*struttura* <classe vincolata> <boundary str>
<tipo vincolo str> <classi vincolanti> <boundary str>
<classi vincolanti> → <classe vincolante> | (<lista classi vincolanti>)
<lista classi vincolanti> → <classe vincolante> , <lista classi vincolanti> |
<classe vincolante>

e la seguente variante alla rappresentazione grafica del medesimo vincolo:



3.7 Attributi a tratti e a sottoaree

L'attributo a tratti (a sottoaree) è una particolare tipologia di attributo che consente di rappresentare le informazioni che hanno la caratteristica di variare sulla superficie (sottoaree) o sulla linea (tratti) che rappresenta l'attributo geometrico di una certa classe. L'attributo a tratti o a sottoaree può quindi essere aggiunto ad una classe solo se questa già presenta un attributo geometrico di tipo GU_CXCurve2D, GU_CXCurve3D (o GU_CPCurve2D, GU_CPCurve3D) o di tipo GU_CXSurface2D (o GU_CXSurface3D) rispettivamente.

La definizione generale per questo nuovo costrutto può essere data come segue: l'attributo a tratti (a sottoaree) è una funzione che, dato un oggetto istanza della classe, per ogni punto contenuto nell'attributo geometrico di tale istanza restituisce un valore del dominio dell'attributo: tale valore può quindi cambiare da punto a punto.

Definizione 66. Attributo a tratti (a sottoaree)

Un attributo a tratti (a sottoaree) di dominio D associato all'attributo geometrico g di una classe C definisce per ogni istanza c di C la seguente funzione:

$$f_c: PointSet(c.g) \rightarrow D$$

dove: il dominio D può essere un qualsiasi dominio di base o un dominio enumerato o un dominio enumerato gerarchico e la funzione PointSet(x) restituisce l'insieme di punti dello spazio di riferimento che corrisponde al valore geometrico x.

La definizione data ipotizza la possibilità di avere un valore diverso per ogni punto che appartiene al valore di un attributo geometrico; tuttavia, nella rappresentazione dei dati in un sistema reale la funzione f_c viene semplificata, raggruppando i punti in zone dotate di valore omogeneo e memorizzando solo tali zone, dette tratti (o sottoaree), e il valore corrispondente della funzione in quella zona.

Gli attributi a tratti possono essere definiti in GeoUML a due diversi livelli di astrazione: un livello più astratto, che non entra nel merito di alcune scelte di rappresentazione, e un livello meno astratto, che implica alcune scelte più definite. I due livelli sono consistenti, e quindi una definizione iniziale più astratta può essere successivamente trasformata in una più concreta.

Al livello meno astratto gli attributi a tratti si distinguono in:

- ***attributi a tratti strutturali***: in questo caso, nell'ambito della classe su cui si vogliono specificare attributi a tratti, viene generata la geometria di ogni tratto omogeneo rispetto agli attributi a tratti specificati;
- ***attributi a tratti dinamici***: in questo caso, nell'ambito della classe su cui si vogliono specificare attributi a tratti, viene generato un **sistema di ascisse curvilinee** rispetto al quale è possibile specificare i valori degli attributi a tratti come eventi lineari o eventi puntiformi.

Per gli attributi a sottoaree invece esiste un unico livello di definizione, in quanto le sottoaree sono sempre rapresentate tramite elementi geometrici, cioè in forma strutturale. Tuttavia, è possibile specificare a livello astratto anche le sottoaree; ciò corrisponde, per quanto riguarda la semantica della specifica, alla forma strutturale, ma senza precisare quante classi vengono generate per rappresentare l'insieme di attributi a sottoaree definito sulla classe principale.

Gli attributi a tratti (o a sottoaree), nelle loro varianti più concrete, sono rappresentabili in GeoUML utilizzando solo i costrutti di base e alcuni vincoli di struttura. Possono essere quindi considerati un costrutto derivato rispetto ai costrutti di base e ai vincoli di struttura.

Si presentano nelle seguenti sezioni i costrutti per la specifica ddegli attributi a tratti nelle loro varie forme e degli attributi a sottoaree mostrandone la semantica, vale a dire, lo schema con costrutti di base e vincoli di struttura a cui corrispondono.

3.7.1 Attributi a tratti o a sottoaree (definizione astratta)

Per definire un attributo a tratti o a sottoaree è sufficiente aggiungere, dopo la definizione di un attributo normale, la dizione *aTratti su* (oppure solamente *aTratti*) o *aSottoaree su* (oppure solamente *aSottoaree*) seguita dal nome dell'attributo geometrico sul quale si vogliono determinare i tratti o le sottoareee.
Ad esempio, la seguente definizione associa alle strade un attributo di sede che è a tratti sul percorso.

classe strada
    attributi: percorso: GU_CPCurve;
        ...altri attributi ...
        sede: TIPO_SEDE *aTratti su* percorso;
dominio TIPO_SEDE (....)

In alcuni casi l'attributo non è definito su interi tratti lineari ma solamente su singoli punti; in questo caso usiamo la parola chiave *aPunti su* (oppure solamente *aPunti*), come nel seguente esempio:

classe strada
    attributi: percorso: GU_CPCurve;
        ...altri attributi ...
        evento: TIPO_EVENTO *aPunti su* percorso;
dominio TIPO_EVENTO(...)

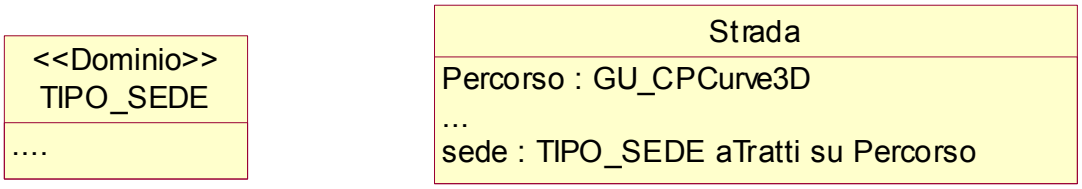
La rappresentazione testuale di un attributo viene estesa come mostrato di seguito:

Rappresentazione testuale (le regole fuori dal riquadro sono già state presentate in precedenza)

<attributo geo> → <chiave> <nome attributo> <cardinalità> **:** <dominio attributo> <a tratti>
<a tratti> → *aTratti su* <id attributo geo> | *aTratti* <id attributo geo> |
aSottoaree su <id attributo geo> | *aSottoaree* <id attributo geo> |
aPunti su <id attributo geo> | *aPunti* <id attributo geo> | ε

<chiave> → ε | *PK*
<nome attributo> → <identificatore>
<id attributo geo> → <identificatore>

La rappresentazione grafica recepisce nella rappresentazione degli attributi a tratti della classe la rappresentazione testuale sopra riportata, come mostrato nel seguente diagramma.


<<Dominio>>
TIPO_SEDE
....
Strada
Percorso : GU_CPCurve3D
...
sede : TIPO_SEDE aTratti su Percorso

3.7.2 Attributi a Tratti Strutturali

In questa rappresentazione del costrutto attributo a tratti ogni tratto viene esplicitamente rappresentato come oggetto di una classe ausiliaria e legato, attraverso una relazione e un vincolo di struttura *qdj-appartiene*, alla classe principale che contiene l'attributo geometrico per il quale si sta definendo l'attributo a tratti. Quindi si propone il seguente schema GeoUML per la rappresentazione dell'attributo a tratti strutturale (nello schema le stringhe di caratteri tra parentesi acute "<esempio>" vanno cosiderate come sostituibili da identificatori e non come simboli non terminali della grammatica; la sintassi GeoUML per gli attributi a tratti strutturali viene specificata nella successiva sezione "Rappresentazione testuale"):

classe <Nome classe principale>
attributi: <attributo geometrico>: <dominio geo lineare>;
...altri attributi ... .
classe <Nome classe ausiliaria>
attributi: <attributo a tratti>: <dominio attributo a tratti>; ..., <altri attributi a tratti>;
geometria : <dominio geo tratti>
ruoli: <ruolo classe principale>[1..1]: <Nome classe principale>
inverso <ruolo tratti>[0..*]; ... .
struttura: <Nome classe ausiliaria>.*tratti* *qdj-appartiene*
<Nome classe ausiliaria>.<ruolo classe principale>.<attributo geometrico>

dove:

- <dominio geo lineare> può essere: GU_CXCurve2D o GU_CNCurve2D o GU_CPCurve2D o GU_CXCurve3D o GU_CNCurve2D o GU_CPCurve3D.
- <dominio geo tratti> può essere: GU_CXCurve2D o GU_CXCurve3D con dimensione consistente con il dominio assunto dall'attributo geometrico della classe principale.

Per facilitare il progettista, si introduce un sintassi abbreviata per la specifica di un attributo a tratti strutturale; tale sintassi consente di specificare solo la classe ausiliaria etichettata con la parola chiave *geometria* e di precisare solo il legame tra la classe ausiliaria e la classe principale.

Rappresentazione testuale

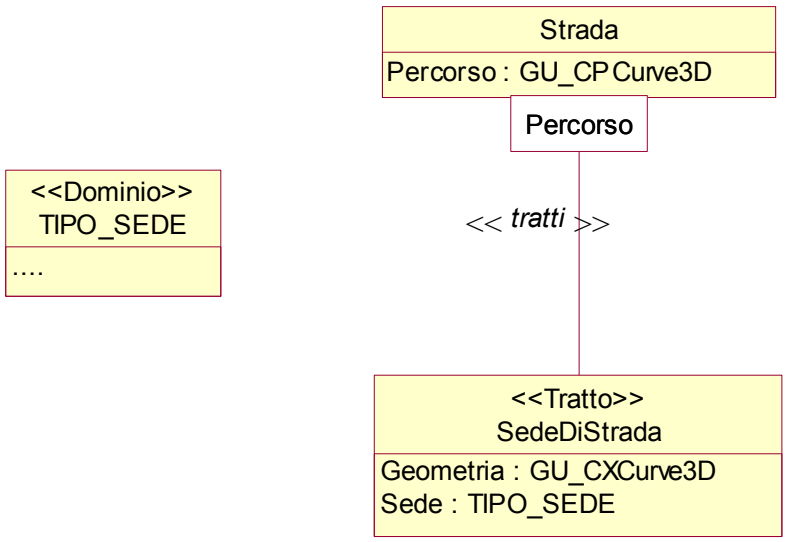
<classe tratto> → *tratto* <classe ausiliaria>
di <classe principale>**.**<attributo geo classe principale>
<proprietà classe ausiliaria>**.**
<classe ausiliaria> → <identificatore classe>
<classe principale> → <identificatore>
<attributo geo classe principale> → <identificatore>
<proprietà classe ausiliaria> → *attributi:* <lista attributi alfa>
<lista attributi alfa> → <attributo alfa>
<lista attributi alfa> → <attributo alfa> **;** <lista attributi alfa>

Ad esempio:

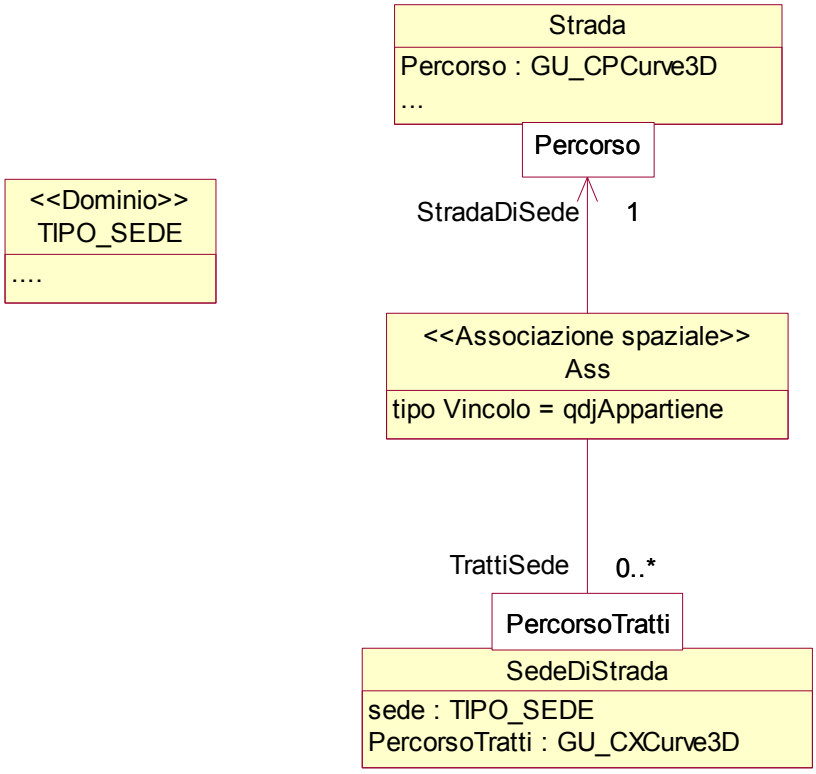
classe strada
attributi: codice: string;
percorso: GU_CPCurve.
tratto SedeDiStrada *di* strada.percorso

attributi: sede: TIPO_SEDE.

Rappresentazione grafica



In questo esempio la rappresentazione esplicita dei vincoli avrebbe prodotto la seguente rappresentazione grafica:



3.7.3 Attributi a Sottoaree strutturali

Il costrutto attributo a sottoaree è rappresentabile in GeoUML solo in modo strutturale. Analogamente all'attributo a tratti strutturale su geometrie lineari, anche per l'attributo a sottoaree ogni sottoarea viene esplicitamente rappresentata come oggetto di una classe ausiliaria e legata, attraverso una relazione e un vincolo di struttura *qdj-appartiene*, alla classe principale che contiene

l'attributo geometrico per il quale si sta definendo l'attributo a sottoaree. Quindi si propone il seguente schema GeoUML per la rappresentazione dell'attributo a sottoaree strutturale (nello schema le stringhe di caratteri tra parentesi acute "<esempio>" vanno cosiderate come sostituibili da identificatori e non come simboli non terminali della grammatica; la sintassi GeoUML per gli attributi a sottoaree strutturali viene specificata nella successiva sezione "Rappresentazione testuale"):

classe <Nome classe principale>
attributi: <attributo geometrico>: <dominio geo areale>;
...altri attributi ... .
classe <Nome classe ausiliaria>
attributi: <attributo a sottoaree>: <dominio attributo a sottoaree>; ...
<altri attributi a sottoaree> ;
geometria: GU_CXSurface2D
ruoli: <ruolo classe principale>[1..1]: <Nome classe principale>
inverso <ruolo sottoaree>[0..*]; ... .
struttura: <Nome classe ausiliaria>.*sottoaree qdj-appartiene*
<Nome classe ausiliaria>.<ruolo classe principale>.<attributo geometrico>

dove: <dominio geo areale> può essere: GU_CXSurface2D o GU_CPSurface2D.

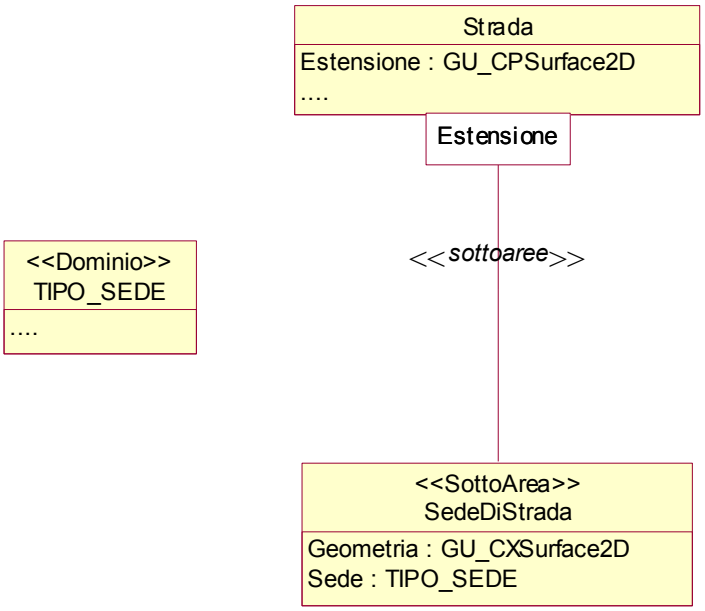
Per facilitare il progettista, si introduce un sintassi abbreviata per la specifica di un attributo a sottoaree strutturale; tale sintassi consente di specificare solo la classe ausiliaria etichettata con la parola chiave *geometria* e di precisare solo il legame tra la classe ausiliaria e la classe principale.

Rappresentazione testuale (le regole fuori dal riquadro sono già state presentate in precedenza)

<classe sottoarea> → *sottoarea* <classe ausiliaria>
di <classe principale>**.**<attributo geo classe principale>
<proprietà classe ausiliaria>**.**

<classe ausiliaria> → <identificatore classe>
<classe principale> → <identificatore>
<attributo geo classe principale> → <identificatore>
<proprietà classe ausiliaria> → *attributi:* <lista attributi alfa>
<lista attributi alfa> → <attributo alfa>
<lista attributi alfa> → <attributo alfa> **;** <lista attributi alfa>

Rappresentazione grafica



3.7.4 Attributi a tratti dinamici

Gli attributi a tratti dinamici svolgono una funzione simile a quella degli attributi a tratti strutturali, ma individuano un tratto non come un insieme di primitive geometriche ma come una porzione misurata lungo un percorso lineare attraverso l'introduzione di un ascissa curvilinea. La definizione di un tratto dinamico, detto anche "evento lineare", presuppone di avere un insieme ordinato, anche se non necessariamente connesso, di "segmenti di percorso", ognuno di tipo GU_CPCurve, sul quale viene misurata l'ascissa curvilinea del punto iniziale e di quello finale. Tale insieme di segmenti deve essere contenuto nell'attributo geometrico dell'oggetto al quale l'attributo a tratti si riferisce, ma può non coincidere con esso. La definizione della classe ausiliaria deve includere anche la dichiarazione di un'unità di misura e del metodo di misura, che può assumere i valori *assoluto* oppure *relativo*. Con il metodo assoluto le misure sono prese dall'inizio dell'oggetto geometrico di riferimento, mentre con il metodo relativo le misure possono riferirsi ad altri punti presenti sull'oggetto geometrico.

Quindi si propone il seguente schema GeoUML per la rappresentazione dell'attributo a tratti dinamico con riferimento assoluto (nello schema le stringhe di caratteri tra parentesi acute "<esempio>" vanno cosiderate come sostituibili da identificatori e non come simboli non terminali della grammatica; la sintassi GeoUML per gli attributi a tratti dinamici viene specificata nella successiva sezione "Rappresentazione testuale"):

classe <Nome classe principale>
    attributi: <attributo geo>: <dominio geo lineare>;
        ...altri attributi ...
classe <Nome classe ausiliaria segmenti>
    attributi: geometria: GU_CPCurve;
        lunghezza: real;
        unitàDiMisura: (centimetri, metri, chilometri);
    ruoli: <ruolo classe principale> [1..*] : <Nome classe principale>
        inverso <ruolo segmento> [0..*] *ordinati*;
struttura: <Nome classe ausiliaria segmenti>.geometria

qdj-appartiene <Nome classe ausiliaria segmenti>.<ruolo classe principale>.<attributo geo>

classe <Nome classe evento lineare>
attributi: <evento lineare>: <dominio evento lineare>; ... <altri eventi lineari>;.
MisuraInizio: real;
MisuraFine: real
ruoli: <ruolo classe principale>[1..1]: <Nome classe principale>
inverso <ruolo evento lineare>[0..*];

dove <dominio geo lineare> può essere: GU_CXCurve2D o GU_CNCurve2D o GU_CPCurve2D o GU_CXCurve3D o GU_CNCurve2D o GU_CPCurve3D.

Si propone invece il seguente schema GeoUML per la rappresentazione dell'attributo a tratti dinamico con riferimento relativo; si noti che in questo caso è presente una classe per rappresentare i punti rispetto ai quali viene specificata la misura:

classe <Nome classe principale>
attributi: <attributo geo>: <dominio geo lineare>;
...altri attributi ...
classe <Nome classe referenti>
attributi: <attributo geo puntiforme>: <dominio geo puntiforme>;
...altri attributi ...
struttura <Nome classe referenti>.<attributo geo puntiforme>
appartiene <Nome classe principale>.<attributo geo>

classe <Nome classe ausiliaria segmenti>
attributi: geometria: GU_CPCurve;
lunghezza: real;
unitàDiMisura: (centimetri, metri, chilometri);
ruoli: <ruolo classe principale> [1..*] : <Nome classe principale>
inverso <ruolo segmento> [0..*] *ordinati*;
struttura: <Nome classe ausiliaria segmenti>.geometria
qdj-appartiene <Nome classe ausiliaria segmenti>.<ruolo classe principale>.<attributo geo>

classe <Nome classe evento lineare>
attributi: <evento lineare>: <dominio evento lineare>; ... <altri eventi lineari>;.
MisuraInizio: real;
MisuraFine: real
ruoli: <ruolo classe referenti>[1..1]: <Nome classe referenti>
inverso <ruolo evento lineare>[0..*];

Per facilitare il progettista, si introduce un sintassi abbreviata per la specifica di un attributo a tratti dinamico; tale sintassi consente di specificare solo la classe ausiliaria che rappresenta gli eventi lineari etichettata con la parola chiave *eventoLineare* e di precisare solo il legame tra la classe ausiliaria e la classe principale.

Rappresentazione testuale

<classe evento lineare> → *eventoLineare*<classe ausiliaria>
di <classe principale>**.**<attributo geo classe principale>
unitàMisura <unità> *metodo* <metodo riferimento lineare>
<proprietà classe evento>**.**
<unità>→ *centimetri* | *metri* | *chilometri* | *decametri* | *miglia*
<metodo riferimento lineare> → *relativo a* <classe dei riferimenti> | *assoluto*
<classe dei riferimenti> → <identificatore>
<proprietà classe evento> → *attributi:* <lista attributi alfa>
<classe ausiliaria> → <identificatore classe>
<classe principale> → <identificatore>
<attributo geo classe principale> → <identificatore>
<lista attributi alfa> → <attributo alfa>
<lista attributi alfa> → <attributo alfa> **;** <lista attributi alfa>

Applicando queste regole allo stesso esempio utilizzato per la dichiarazione dell'attributo a tratti "sede" in forma strutturale, otteniamo la seguente dichiarazione di attributo dinamico o evento lineare con metodo assoluto:

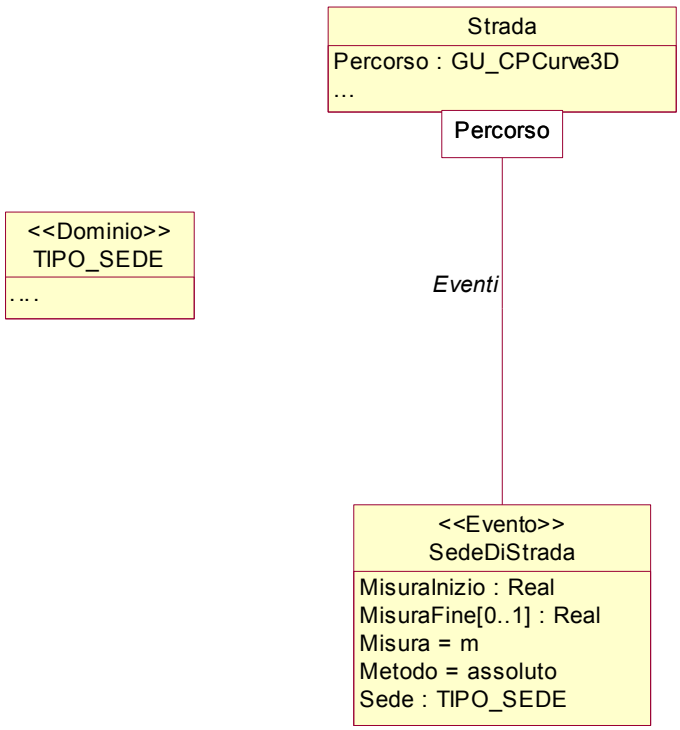
classe strada
attributi: percorso: GU_CPCurve;
codice: string.
eventoLineare SedeDiStrada *di* strada.percorso
unitàMisura metri *metodo* *assoluto*;
attributi: sede: TIPO_SEDE.

E' possibile anche rappresentare eventi puntiformi sempre con riferimento al sistema di ascissa curvilinea definito su un attributo geometrico lineare. La struttura delle classi e come quella per gli eventi lineari, l'unica differenza è la mancanza dell'attributo MisuraFine nella classe ausiliaria che rappresenta l'evento puntiforme. La sintassi per la rappresentazione testuale di eventi puntiformi è la seguente:

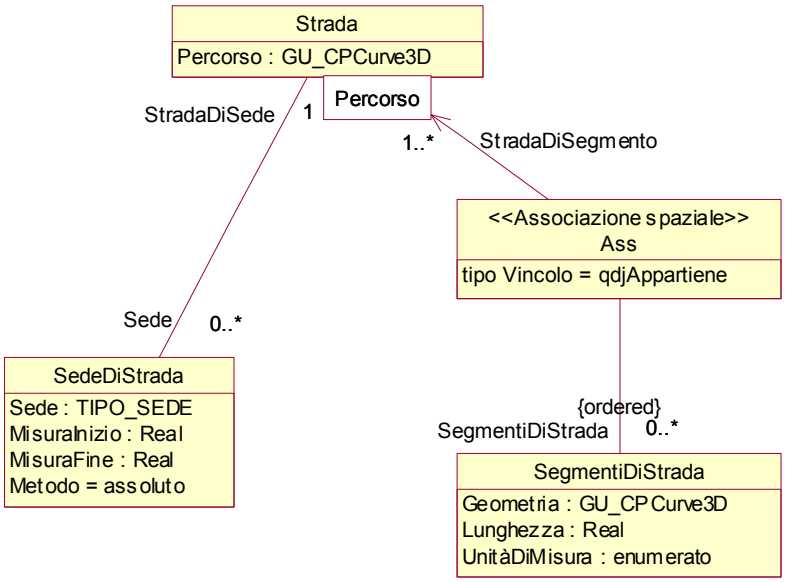
Rappresentazione testuale

<classe evento puntiforme> → *eventoPuntiforme* <classe ausiliaria>
di <classe principale>**.**<attributo geo classe principale>
unitàMisura <unità> *metodo* <metodo riferimento lineare>
<proprietà classe evento>**.**

Rappresentazione grafica



In questo esempio la rappresentazione esplicita dei vincoli e delle classi ausiliarie porterebbe al seguente schema:



Per gli eventi puntiformi la rappresentazione grafica è simile a quella vista per gli eventi lineari; l'unica differenza si presenta negli attributi della classe ausiliaria: vale a dire, invece degli attributi MisuraInizio e MisuraFine si ha un unico attributo MisuraPosizione.

Un attributo a tratti può essere definito sul boundary di un attributo geometrico (di tipo GM_Surface o GM_ComplexSurface). In tal caso i vincoli "Appartiene" devono essere riferiti al boundary della classe principale (vedi paragrafo 3.6.5)

Appendice 1 Cenni al linguaggio Object Constraint Language (OCL)

1.1 Introduzione

In questa appendice vengono presentate le caratteristiche fondamentali dell'Object Constraint Language (OCL), ossia del linguaggio formale utilizzato per esprimere vincoli nel linguaggio UML, con particolare riferimento alle formule utilizzate in questo documento. Non si vuole quindi sostituire il manuale OCL, al quale si rimanda per tutti i dettagli, ma semplicemente fornire un'introduzione al linguaggio che faciliti una lettura del documento.

OCL può essere utilizzato con scopi differenti: per specificare vincoli invarianti su classi e tipi all'interno del modello delle classi, per individuare condizioni (e tipi) invarianti riguardanti gli *stereotipi (stereotypes)*, per descrivere pre e post condizioni sulle operazioni e sui metodi, per descrivere condizioni di guardia e per esprimere vincoli sulle operazioni. In genere ogni espressione OCL è introdotta per mezzo della parola chiave *context* seguita dal nome del tipo (classe o associazione) su cui è posta la condizione invariante dell'espressione OCL. Quando il vincolo OCL è posto nel diagramma UML, esso è graficamente collegato a tale tipo.

L'esempio di schema UML di figura A1.1 verrà utilizzato nel seguito per illustrare con esempi le caratteristiche di OCL.

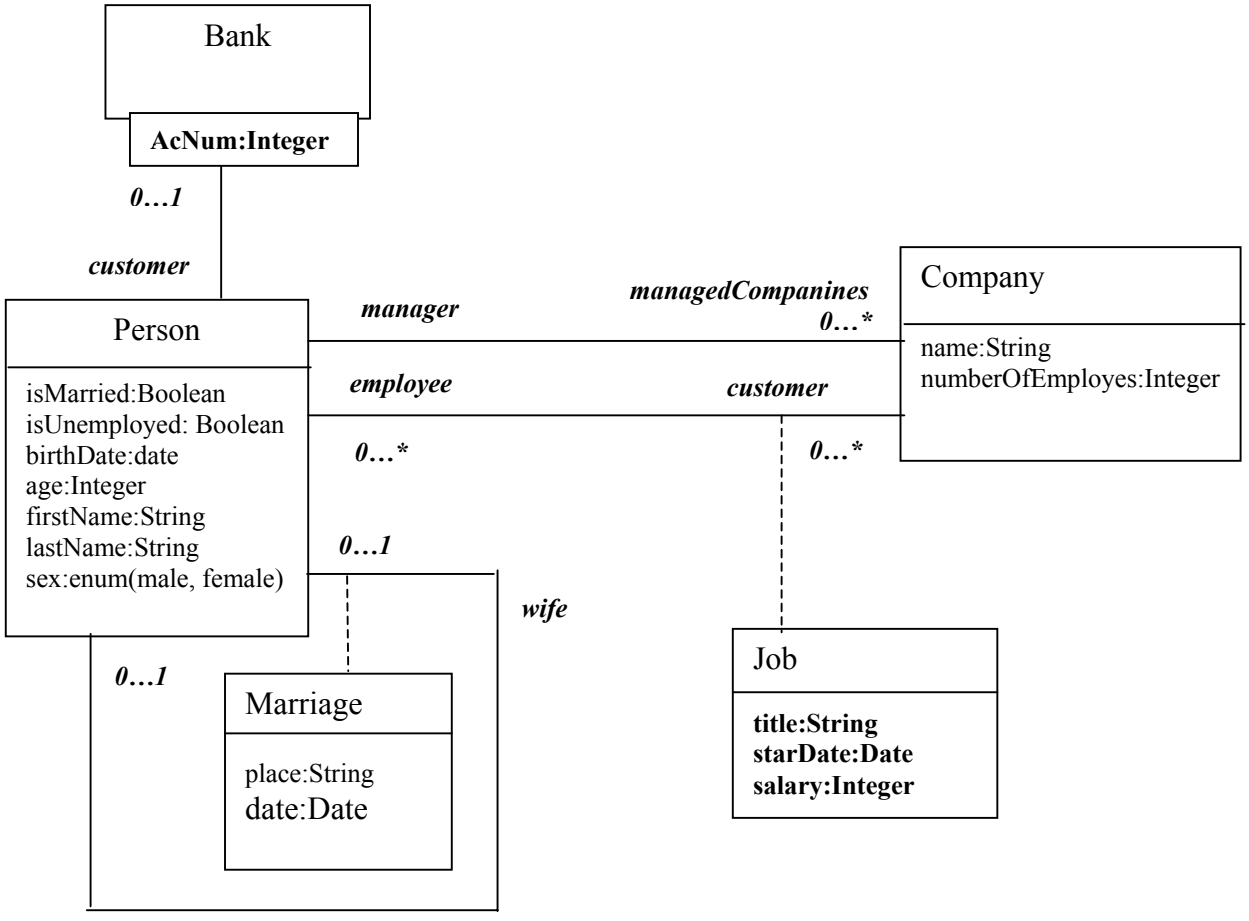


Figura A1.1: esempio per OCL (estratto dal documento OMG per la versione UML 2.0).

1.2 Elementi fondamentali del linguaggio

Gli elementi fondamentali del linguaggio permettono di definire il contesto al quale si applica il vincolo, di riferirsi ad un elemento di tale contesto, di accedere alle proprietà di tale elemento, di dichiarare che il vincolo è un invariante, di accedere agli elementi collegati a un oggetto navigando un'associazione, di selezionare tutti gli oggetti di una classe.

- *context (specifica del contesto):* il contesto di un espressione OCL per un modello UML può essere specificato per mezzo di una *dichiarazione del contesto* posta all'inizio dell'espressione OCL
- *self: l*a parola riservata *self* è utilizzata per riferirsi a una generica istanza del contesto. Ad esempio se il contesto è costituito dalla classe *Company,* allora la parola riservata *self* è utilizzata per riferirsi ad una generica istanza di *Company*.
- *invarianti:* un'espressione OCL è un invariante per un tipo T se tale espressione deve sempre essere verificata per ogni istanza del tipo T. La parola riservata *inv* definisce un vincolo di natura invariante. Il seguente esempio stabilisce che per ogni istanza della classe *Company* il numero degli impiegati dovrà sempre essere maggiore di cinquanta (vale la pena di osservare che quando il contesto in cui si opera è esplicitamente chiaro la parola riservata *self* può essere omessa dalla dichiarazione del vincolo):

  context Company inv:

  self.numberOfEmployees > 50.

Le espressioni OCL possono essere indirizzate a tipi, classi, interfacce ed associazioni. E' possibile utilizzare all'interno di tali espressioni attributi, metodi, operazioni ed estremi delle associazioni (ruoli); l'insieme di questi elemnti è indicato con il termine genrico di *proprietà.* Il valore di una proprietà di un oggetto è definito specificando il nome dell'oggetto stesso mediante la parola riservata *self*, seguito da un punto e dal nome della proprietà desiderata, ad esempio *self.property*.

- *attributi:* l' attributo *age* dell'oggetto *Person* può essere selezionato mediante il costrutto *self.age*; in questo modo si otterrà il valore dell'attributo *age* per una particolare istanza di *Person* identificata mediante la parola riservata *self*. Con la seguente espressione si dichiara che il valore dell'attributo *age* di ogni istanza dell'oggetto *Person* deve sempre essere maggiore di zero:

  context Person inv:

  self.age > 0.

- *lati di associazione e navigabilità:* un'associazione, presente in un diagramma delle classi, può essere navigata servendosi del costrutto *object.rolename*, l'associazione è percorsa servendosi dell'oggetto posto sul lato opposto a quello di partenza. Il valore dell'espressione appena introdotta è costituito dall'insieme degli oggetti dotati del *rolename* indicato. Si osservi il seguente esempio:

  context Company

  inv: self.manager.isUnemployed = false

  inv: self.employee → notEmpty

per come è disposta la cardinalità all'interno dello schema il primo invariante individua un solo oggetto di tipo *Person*; il secondo invariante dovrà invece essere valutato in un insieme (set) di

oggetti di tipo *Person*. In genere per accedere agli insiemi si utilizza il simbolo → seguito dal nome della proprietà che si vuole introdurre. Se non esiste un *rolename* si può utilizzare quello della classe con la lettera minuscola. Un'altra considerazione merita il caso in cui la cardinalità dell'oggetto sia opzionale (0…1), in questo caso conviene utilizzare un'espressione che permetta di operare quando la cardinalità risulti unitaria:

context Person inv:

self.wife → notEmpty implies self.wife.sex = #female.

Ovviamente le espressioni possono sempre essere combinate tra loro per ottenere condizioni più complesse e complete.

- - *allIstances*: *AllIstances* si riferisce all'insieme delle istanze di un tipo che esistono nel momento in cui l'espressione è valutata (mentre self si riferisce a una generica istanza). Si osservi il seguente esempio:

context Person inv:

Person.allIstances → forAll (p1,p2 \| p1<>p2 implies p1.name <> p2.name)

Esso permette di stabilire che tutte le istanze di tipo *Person* debbano avere un nome unico (per l'impiego di *forAll* si veda il prossimo paragrafo).

1.3 Collezioni e quantificazione.

Come mostrato dal precedente esempio, le espressioni OCL possono riferirsi, oltre che a singoli oggetti, anche ad insiemi.

Il tipo *Collection* è un tipo predefinito di OCL; *Collection* è un tipo astratto dotato dei sottotipi istanziabili *Set, Bag e Sequence*. Un *Set* è un insieme matematico, privo di elementi duplicati; un *Bag* è un insieme di elementi in cui sono ammessi duplicati; una *Sequence* è una collezione ordinata di elementi che ammette duplicati. Gli elementi delle collezioni sono indicati all'interno di parentesi graffe. Ad esempio:

Sequence { 1,2,4,5,6,3,4}

Sequence { 1,....,10}

Set { 1, 2, 4 ,5 ,6 }

OCL definisce alcune operazioni di base sulle collezioni, le principali sono:

- operazione di ***Select***: in alcuni casi un'espressione che utilizza operazioni e/o navigatori si riferisce ad una collezione od in particolare ad un sottoinsieme di elementi della collezione. OCL permette di selezionare specifici elementi di una collezione mediante il costrutto *collection → select (.....)*. All'interno del *Select* può comparire un'espressione booleana generando un'espressione del tipo *collection → select (boolean expression)*; saranno quindi prelevati tutti gli elementi della collezione per cui l'espressione introdotta abbia valore *true*. Si osservi il seguente esempio:

context Company inv:

self.employee → select (age > 50) → notEmpty

In questo caso si preleveranno dalla collezione *Set (Person)*, che identifica il tipo di *self.employee*, tutti i dipendenti che avranno *age > 50*.

Nell'esempio precedente non è possibile riferirsi in modo esplicito agli oggetti di tipo *Person*, ma è stato fatto un riferimento ad una loro proprietà; per individuare in modo specifico un singolo oggetto di tipo *Person* bisogna utilizzare il costrutto *collection → select (v | boolean expression with v)*. La variabile v prende il nome di *iterator*, v può essere utilizzata per riferirsi in modo specifico agli oggetti della collezione. La condizione espressa nell'esempio precedente assumerà la forma:

context Company inv:

self.employee → select (p | p.age > 50) → notEmpty

Esiste un ulteriore estensione dell'espressione *Select*, che permette di dichiarare un tipo per la variabile v; questo consente di specificare che gli elementi della collezione saranno del tipo associato all'*iterator*, la forma del costrutto diventa *collection → select (v:Type | boolean expression with v)*. L'esempio assume la forma:

context Company inv:

self.employee → select (p:Person | p.age > 50) → notEmpty

- operazione di ***Reject***: l'operazione *Reject* è sintatticamente identica all'operazione *Select*, in questo caso sono prelevati gli elementi della collezione per cui la condizione è valutata *false.*

- operazione di ***Collect***: le operazioni *Select* e *Reject* permettono di generare un sottoinsieme della collezione di partenza, in particolare se la collezione di partenza avrà elementi di tipo *Person* anche la nuova collezione generata avrà elementi di tipo *Person*. Con l'operazione *Collect* è possibile ottenere una nuova collezione contenente elementi diversi da quelli presenti nella collezione di partenza, ad esempio prendendo come collezione di partenza la collezione degli oggetti di tipo *Person* è possibile utilizzare l'operazione *Collect* per ottenere l'insieme delle *birthDates* di tutti gli oggetti di tipo *Persons*. La sintassi è analoga a quella degli operatori *Select* e *Reject.*

collection → collect (expression)

collection → collect (v | expression with v)

collection → collect (v:Type | expression with v)

E possibile ottenere la collezione delle birthDates con una qualsiasi delle seguenti espressioni:

self.employee → collect (birthDate)

self.employee → collect (person | person.birthDate)

self.employee → collect (person : Person | person.birthDate)

La collezione ottenuta apparterrà al tipo Bag.

- operazione di ***ForAll***: alcune situazioni portano a dover introdurre un vincolo per tutti gli elementi della collezione, in questo caso è necessario utilizzare l'operazione *ForAll*. La sintassi è analoga a quella delle operazioni *Select* e *Reject.*

collection → forAll (boolean expression)

collection → forAll (v | boolean expression with v)

collection → forAll (v:Type | boolean expression with v)

Il risultato di tale operazione è un espressione booleana, essa sarà *true* se la condizione introdotta è soddisfatta da tutti gli elementi della collezione, sarà *false* se almeno un elemento della collezione non soddisfa la condizione assegnata. All' interno di tale operazione è possibile utilizzare più di un *iterator*, in questo caso il comportamento del costrutto è analogo a quello del prodotto cartesiano; l'esempio seguente mostra un espressione con due *iterators*:

context Company inv:

self.employee → forAll (e1, e2 : Person | e1<> e2 implies e1.forename <> e2.forename)

- operazione di ***Exists***: in alcuni casi è necessario sapere se vi sia almeno un elemento della collezione che soddisfi un vincolo assegnato; l'operatore *Exists* permette di specificare un espressione booleana che deve essere verificata per almeno un elemento di una collezione. Il risultato di tale operazione è un valore booleano, *true* se l'espressione booleana è valida per almeno un elemento della collezione, false altrimenti. La sintassi è analoga a quella delle operazioni *Select* e *Reject.*

collection → exists (boolean expression)

collection → exists (v | boolean expression with v)

collection → exists (v:Type | boolean expression with v)

Il seguente esempio mostra una condizione booleana valutata sul *firstName* di tutti gli elementi della collezione di tipo *Person,* l'operatore darà risultato *true* se almeno un oggetto della collezione avrà come *firstName* il valore '*Jack*'.

context Company inv:

self.employee → exists (p : Person | p.firstName = 'Jack')

- operazione di ***Iterate***: è un operatore assai complesso ma di utilizzo estremamente generale. Si basa sul concetto di accumulazione; un valore è calcolato, mediante accumulazione, iterando il valore stesso all'interno di una collezione. La sintassi dell'operatore è la seguente:

  collection → iterate(elem:Type, acc:Type; Type =<expression> | expression with elem and acc). La variabile *elem* è l'iteratore, la variabile *acc* è l'accumulatore, l'accumulatore ha come valore iniziale <expression>. *Elem* è fatto iterare all'interno della collezione è l'espressione con *elem* ed *acc* è valutata per ogni elemento (*elem*) della collezione.Il valore ottenuto per la valutazione dell'espressione, con *elem* ed *acc*, è assegnato ad *acc*; in questo modo il valore di *acc* è ottenuto mediante l'iterazione sulla collezione.

1.4 Valori e tipi

In OCL vi sono tipi base e valori predefiniti a disposizione dell'utente. In genere sarà sempre possibile definire ulteriori tipi fruibili all'interno delle espressioni OCL. I tipi aggiuntivi ai quali sarà possibile riferirsi dovranno comunque sempre essere definiti all'interno del meta-modello UML.

La tabella 1 mostra i tipi di base e le operazioni su di essi eseguibili.

Tipi Base.	***Valori.***	***Operazioni.***
Boolean	**True, false**	**and, or, xor, not, implies, if..then..else**
Integer	**1, -5, 2, 34,…**	***, +, -, /, floor**

Real	**1.5, 3.14, …**	***, +, -, /, abs**
String	**“ essere o non essere”**	**toUpper, concat**

Tabella 1: tipi base per OCL (estratto dal documento OMG per la versione UML 2.0).

E’ possibile definire ulteriori tipi e costrutti.

1.5 Navigabilità sulle associazioni

E’ possibile specificare anche navigabilità all’interno delle classi di associazione utilizzando il nome della classe di associazione con la lettera iniziale minuscola. Il seguente esempio utilizza la classe di associazione *job*:

context** Person **inv:

***self**.job*

Nel caso di un’associazione ricorsiva, ossia di una classe in associazione con se stessa il solo riferimento al nome della classe di associazione non è sufficiente, bisogna specificare anche la direzione di percorrenza. Si osservi il seguente esempio:

context** Person **inv:

***self**.marriage[husband]*

In questo caso risulta chiaro che ci si riferisca ad un *Marriage* relativo all’eventuale *husband*; altrimenti sarebbe stato possibile riferirsi indistintamente al *Marriage* di *husband* o di *wife*. E’ permesso navigare dalla classe di associazione verso gli oggetti che partecipano alla relazione, questo può essere fatto utilizzando la notazione prevista per un singolo oggetto ed i nomi di ruolo assegnati ai lati di un’associazione. Si osservi il seguente esempio:

context** Job **inv:

***self**.employer.numberOfEmployess > 1*

***self**.employee.age > 21*

- *navigazione attraverso associazioni qualificate:* le associazioni qualificate utilizzano uno o più attributi dell’associazione per selezionare elementi presenti sul lato opposto dell’associazione. Si osservi il seguente esempio:

context** Bank **inv:

***self**.customer[8764]*

Il risultato di tale operazione permette di individuare l’istanza di *Person* che abbia come *accountNumber* il numero intero *8764*.

- *accesso alle proprietà dei supertipi:* è sempre possibile ridefinire le proprietà all’interno di un dato tipo a patto che questo sia un sottotipo valido di un supertipo assegnato; si supponga di avere una classe *B* sottotipo di una classe *A*, si supponga inoltre che *A* o *B* condividano una proprietà *Pab*; il seguente esempio mostra come accedere prima alla proprietà *Pab* della classe *A* e successivamente alla proprietà *Pab* della classe *B*:

context** B **inv:

self.oclAsType(A).Pab

self.Pab

- *operazioni predefinite:* esistono diverse operazioni predefinite per i tipi presenti in OCL:

-*oclIsTypeOf (t : OclType) : Boolean*. Il risultato di tale operazione è *true* se il tipo individuato dall'oggetto su cui è invocato è lo stesso di quello individuato da t.

-*oclIsKindOf (t : OclType) : Boolean*. Tale operazione determina se t sia o meno il supertipo di un oggetto.

-*oclInState (s : OclState) : Boolean.* Il risultato di tale operazione è *true* se l'oggetto si trova nello stato s.

Vale la pena di ricordare infine che tutte le operazioni sin qui descritte sono relative ad istanze di classi. I tipi delle classi possono essere sia quelli predefiniti in OCL sia quelli definiti all'interno del modello.

Appendice 2 Sintassi della rappresentazione testuale del GeoUML

Si riporta di seguito la rappresentazione formale della sintassi del linguaggio nella forma testuale. Si usa la notazione delle grammatiche BNF indicando i simboli terminali in corsivo (o corsivo e sottolineato). L'assioma è <S>.

Si presentano due versioni della grammatica: la prima versione è quella di maggior leggibilità per l'utente, la seconda è equivalente alla prima (vale a dire produce lo stesso linguaggio) ed ha in più la caratteristica di essere una grammatica della classe LL(1) (Left, Look ahead 1) e consente quindi la costruzione automatica di un analizzatore sintattico discendente.

VERSIONE 1

Regole generali

<S> → <classe>
<S> → <S> <elemento>
<elemento> → <classe> | <dominio> | <associazione> | <vincolo topologico> |
<vincolo di struttura> | <strato topologico> | <classe sottoarea> | <classe tratto> |
<classe evento lineare> | <classe evento puntiforme>

Regole sulle classi

<classe> → *classe* <identificatore classe> <classificazione completa> <sottoclasse di>
<proprietà della classe> **.**

<identificatore classe> → <identificatore> *(* <identificatore abbreviato> *)* | <identificatore>
<identificatore abbreviato> → <identificatore>
<sottoclasse di> → *sottoclasse di* <identificatore altra classe> | ε
<identificatore altra classe> → <identificatore>

<classificazione completa> → <vincoli classificazione> *(* <lista classi figlie> *)* | ε
<vincoli classificazione> → <vincolo classif> | <vincolo classif 1> , <vincolo classif 2>
<vincolo classif> → *complete* | *incomplete* | *disjoint* | *overlapping*
<vincolo classif 1> → *complete* | *incomplete*
<vincolo classif 2> → *disjoint* | *overlapping*
<lista classi figlie> → <classe figlia>
<lista classi figlie> → <classe figlia> , <lista classi figlie>
<classe figlia> → <identificatore>

Regole sulle classi: attributi

<proprietà della classe> → *attributi:* <lista attributi> <ruoli> <vincoli>
<lista attributi> → <attributo>
<lista attributi> → <attributo> **;** <lista attributi>
<attributo> → <attributo alfa> | <attributo geo>
<attributo alfa> → <chiave> <nome attributo> <cardinalità> **:** <dominio alfa>
<attributo geo> → <nome attributo> <cardinalità> **:** <dominio geo> <a tratti>
<chiave> → ε | *PK*
<nome attributo> → <identificatore>

<a tratti> → *aTratti su* <id attributo geo> | *aTratti* <id attributo geo> |
aPunti su <id attributo geo> | *aPunti* <id attributo geo> |
aSottoaree su <id attributo geo> | *aSottoaree* <id attributo geo> | ε
<id attributo geo> → <identificatore>

<dominio alfa> → <identificatore> | <dominio enumerato> | *String* | *Integer* | *Real*
<dominio geo> → *GU_Point2D* | *GU_Point3D* | *GU_CPCurve2D* | *GU_CPCurve3D* |
GU_Ring2D | *GU_Ring3D* | *GU_CPSurface2D* | *GU_CXCurve2D* |
GU_CXCurve3D | *GU_CXRing2D* | *GU_CXRing3D* | *GU_CXSurface2D* |
GU_Complex2D | *GU_Complex2D* | *GU_Aggregate2D* | *GU_Aggregate3D* |
GU_MPoint2D | *GU_MPoint3D* | *GU_MCurve2D* | *GU_MCurve3D* |
GU_MSurface2D | *GU_MRing2D* | *GU_MRing3D* | *GU_CNCurve2D* |
GU_CPSurfaceB3D | *GU_CXSurfaceB3D* | *GU_CNCurve3D*

<dominio enumerato> → *(* <lista valori> *)*
<lista valori> → <valore>
<lista valori> → <valore> **,** <lista valori>
<valore> → <identificatore> | <valore cond>
<valore cond> → <identificatore> <sep> <lista cond opzionale>
<lista cond opzionale> → <attributo cond> *: (* <lista valori> *)*
<lista cond opzionale> → <attributo cond> *: (* <lista valori> *)* <lista cond opzionale>
<attributo cond> → <identificatore>

<cardinalità> → ε
<cardinalità> → *[* <card min>..<card max> *]*
<card min> → *0* | <cifra diversa da zero> <numero>
<numero> → <cifra> <numero> | ε
<card max> → * | <cifra diversa da zero> <numero>

Regole sulle classi: ruoli
<ruoli> → ε
<ruoli> → *; ruoli:* <lista ruoli>
<lista ruoli> → <ruolo>
<lista ruoli> → <ruolo> **;** <lista ruoli>
<ruolo> → <chiave> <ruolo classe associata> <cardinalità> *:* <classe associata>
inverso <ruolo inverso> <cardinalità>
<ruolo> → <chiave> <ruolo classe associata> <cardinalità> *:* <classe associata>
<ruolo classe associata> → <identificatore>
<ruolo inverso> → <identificatore>
<classe associata> → <identificatore>

Regole sulle classi: vincoli
<vincoli> → ε
<vincoli> → *; vincoli :* <lista vincoli>
<lista vincoli> → <vincolo>
<lista vincoli> → <vincolo> **;** <lista vincoli>
<vincolo>→ <vincolo in OCL>

<vincolo in OCL> → "vedi sintassi OCL"

Regole sulle associazioni

<associazione> → *associazione* <id associazione> *(* <prima classe>, <seconda classe> *)*
<proprietà della associazione> **.**
<id associazione> → <identificatore> **(** <identificatore abbreviato> **)** | <identificatore>
<prima classe> → <identificatore>
<seconda classe> → <identificatore>
<proprietà della associazione> → *attributi:* <lista attributi alfa> <vincoli>
<lista attributi alfa> → <attributo alfa>
<lista attributi alfa> → <attributo alfa> **;** <lista attributi alfa>

Regole sui domini

<dominio> → *dominio* <identificatore dominio> **:** <definizione enumerato> **.**

Regole sugli strati topologici

<strato topologico> → *strato* <id strato> <proprietà strato>**.**
<proprietà strato> → *geometria:* <dominio strato> <vincoli>
<id strato> → <identificatore> (<identificatore abbreviato>) | <identificatore>
<dominio strato> → *GU_Complex2D* | *GU_Complex3D* | *GU_CXCurve2D* |
GU_CXCurve3D | *GU_CXSurface2D* | *GU_CNCurve2D* |
GU_CNCurve2D | *GU_CXRing2D* | *GU_CXRing3D*

Regole sugli attributi a tratti strutturali e dinamici
Attributi a tratti strutturali

<classe tratto> → *tratto* <classe ausiliaria>
di <classe principale>**.**<attributo geo classe principale>
<proprietà classe ausiliaria>**.**
<classe ausiliaria> → <identificatore classe>
<classe principale> → <identificatore>
<attributo geo classe principale> → <identificatore>
<proprietà classe ausiliaria> → *attributi:* <lista attributi alfa>
<lista attributi alfa> → <attributo alfa>
<lista attributi alfa> → <attributo alfa> **;** <lista attributi alfa>

Attributi a tratti dinamico

<classe evento lineare> → *eventoLineare*<classe ausiliaria>
di <classe principale>**.**<attributo geo classe principale>
unitàMisura <unità> *metodo* <metodo riferimento lineare>
<proprietà classe evento>**.**
<classe ausiliaria> → <identificatore classe>
<classe principale> → <identificatore>
<attributo geo classe principale> → <identificatore>
<unità>→ *centimetri* | *metri* | *chilometri* | *decametri* | *miglia*
<metodo riferimento lineare> → *relativo a* <classe dei riferimenti> | *assoluto*
<classe dei riferimenti> → <identificatore>
<proprietà classe evento> → *attributi:* <lista attributi alfa>

<classe evento puntiforme> → *eventoPuntiforme* <classe ausiliaria>
di <classe principale>**.**<attributo geo classe principale>
unitàMisura <unità> *metodo* <metodo riferimento lineare>
<proprietà classe evento>**.**

Regole sugli attributi a sottoaree strutturali

<classe sottoarea> → *sottoarea* <classe ausiliaria>
di <classe principale>**.**<attributo geo classe principale>
<proprietà classe ausiliaria>**.**

Regole sui vincoli topologici

<vincolo topologico> → *Vtopo* <classe vincolata> <selezioneX> <boundary>
<lista relazioni> <tipo vincolo>
<classe vincolante> <selezioneYX> <boundary> <or lista vincoli>
<boundary> → ε | *.bnd* | *.pln* | *.bnd.pln* | *.pln.bnd*
<tipo_vincolo> → *esiste* | *union* | *perOgni*
<classe vincolata> → <identificatore>.<identificatore_attr_geo> |
<identificatore>.<identificatore_attr_geo>.*S* |
<identificatore>.<identificatore_attr_geo>.*B*
<classe vincolante> → <identificatore>.<identificatore_attr_geo> |
<identificatore>.<identificatore_attr_geo>.*S* |
<identificatore>.<identificatore_attr_geo>.*B*
<identificatore>.<id ruolo>.<identificatore_attr_geo> |
<identificatore>.<id ruolo>.<identificatore_attr_geo>.*S* |
<identificatore>.<id ruolo>.<identificatore_attr_geo>.*B*
<identificatore_attr_geo> → <identificatore>
<id ruolo> → <identificatore> | <identificatore>::<identificatore>
<lista relazioni> → <relazione topo>
<lista relazioni> → <relazione topo> , <lista relazioni>
<relazione topo> → *DJ* | *TC* | *IN* | *CT* | *EQ* | *CR* | *OV*
<selezioneX> → ε | *(* <qualsiasi espressione boolean in OCL sugli attributi della
classe vincolata X> *)*
<selezioneYX> → ε | *(* <qualsiasi espressione boolean in OCL sugli attributi della
classe vincolante Y e eventualmente della classe vincolata X> *)*
<or lista vincoli> → *or* <lista vincoli dis> | ε
<lista vincoli dis> → <vincolo topologico>
<lista vincoli dis> → <vincolo topologico> **,** <lista vincoli dis>

Regole sui vincoli di struttura

<vincolo di struttura> → *struttura* <classe vincolata> <boundary str>
<tipo vincolo str> <classi vincolanti> <boundary str>
<classi vincolanti> → <classe vincolante> | (<lista classi vincolanti>)
<lista classi vincolanti> → <classe vincolante> , <lista classi vincolanti> | <classe vincolante>
<boundary str> → ε | *.bnd*
<tipo vincolo str> → *appartiene* | *compostoDa* | *appartiene* | *dj-appartiene* | *qdj-appartiene* |
partizionato | *q-partizionato*

Regole sugli identificatori

<identificatore> → <carattere>
<identificatore> → <carattere> <identificatore>
<carattere> → a | ... | z | A | .... | Z | _ | <cifra>
<cifra> → 0 | ... | 9
<sep> → "spazi, tab e caratteri speciali per indicare fine riga"
<cifra diversa da zero> → 1 | ... | 9

VERSIONE 2 (grammatica LL(1))

Regole generali

<S> → <classe> <S'>
<S'> → <elemento> <S'> | ε
<elemento> → <classe> | <dominio> | <associazione> | <vincolo topologico> |
<vincolo di struttura> | <strato topologico> | <classe sottoarea> | <classe tratto> |
<classe evento lineare> | <classe evento puntiforme>

Regole sulle classi

<classe> → *classe* <identificatore classe> <classificazione completa> <sottoclasse di>
<proprietà della classe> **.**

<identificatore classe> → <identificatore> <identificatore abbreviato>
<identificatore abbreviato> → *(* <identificatore> *)* | ε
<sottoclasse di> → *sottoclasse di* <identificatore altra classe> | ε
<identificatore altra classe> → <identificatore>

<classificazione completa> → <vincoli classificazione> *(*<classe figlia> <lista classi figlie> *)* | ε
<vincoli classificazione> → *complete* <vincolo classif 2> | *incomplete* <vincolo classif 2> |
disjoint | *overlapping*
<vincolo classif 2> → , *disjoint* | , *overlapping* | ε
<lista classi figlie> → ε
<lista classi figlie> → , <classe figlia> <lista classi figlie>
<classe figlia> → <identificatore>

Regole sulle classi: attributi

<proprietà della classe> → *attributi:* <attributo> <lista attributi> <ruoli> <vincoli>
<lista attributi> → ε
<lista attributi> → *;* <attributo> <lista attributi>
<attributo> → <attributo chiave> | <attributo non chiave>
<attributo chiave> → *PK* <nome attributo> <cardinalità> *:* <dominio alfa>
<attributo non chiave> → <nome attributo> <cardinalità> *:* <dominio>
<dominio> → <dominio alfa> <a tratti> | <dominio geo>
<nome attributo> → <identificatore>
<a tratti> → *aTratti su* <id attributo geo> | *aTratti* <id attributo geo> |
aSottoaree su <id attributo geo> | *aSottoaree* <id attributo geo> | ε
aPunti su <id attributo geo> | *aPunti* <id attributo geo> | ε
<id attributo geo> → <identificatore>

<dominio alfa> → <identificatore> | <dominio enumerato> | *String* | *Integer* | *Real*
<dominio geo> → *GU_Point2D* | *GU_Point3D* | *GU_CPCurve2D* | *GU_CPCurve3D* |
GU_Ring2D | *GU_Ring3D* | *GU_CPSurface2D* | *GU_CXCurve2D* |
GU_CXCurve3D | *GU_CXRing2D* | *GU_CXRing3D* | *GU_CXSurface2D* |
GU_Complex2D | *GU_Complex2D* | *GU_Aggregate2D* | *GU_Aggregate3D* |
GU_MPoint2D | *GU_MPoint3D* | *GU_MCurve2D* | *GU_MCurve3D* |
GU_MSurface2D | *GU_MRing2D* | *GU_MRing3D* | *GU_CNCurve2D* |
GU_CPSurfaceB3D | *GU_CXSurfaceB3D* | *GU_CNCurve3D*

<dominio enumerato> → *(*<valore> <lista valori> *)*
<lista valori> → ε
<lista valori> → *,* <valore> <lista valori>
<valore> → <identificatore> <valore cond>
<valore cond> → <sep> <attributo cond> *: (* <lista valori> *)* <lista cond opzionale> | ε
<lista cond opzionale> → ε
<lista cond opzionale> → *,* <attributo cond> *: (* <lista valori> *)* <lista cond opzionale>
<attributo cond> → <identificatore>

<cardinalità> → ε
<cardinalità> → *[* <card min>..<card max> *]*
<card min> → *0* | <cifra diversa da zero> <numero>
<numero> → <cifra> <numero> | ε
<card max> → * | <cifra diversa da zero> <numero>

Regole sulle classi: ruoli
<ruoli> → ε
<ruoli> → *;* *ruoli:* <ruolo> <lista ruoli>
<lista ruoli> → ε
<lista ruoli> → *;* <ruolo> <lista ruoli>
<ruolo> → <chiave> <ruolo classe associata> <cardinalità> *:* <classe associata> <inverso>
<inverso> → *inverso* <ruolo inverso> <cardinalità> | ε
<ruolo classe associata> → <identificatore>
<ruolo inverso> → <identificatore>
<classe associata> → <identificatore>
<chiave> → ε | *PK*

Regole sulle classi: vincoli generici OCL
<vincoli> → ε
<vincoli> → *;* *vincoli :* <vincolo> <lista vincoli>
<lista vincoli> → ε
<lista vincoli> → *;* <vincolo> <lista vincoli>
<vincolo>→ <vincolo in OCL>
<vincolo in OCL> → “vedi sintassi OCL”

Regole sulle associazioni
<associazione> → *associazione* <id associazione> *(* <prima classe>, <seconda classe> *)*

<proprietà della associazione> **.**
<id associazione> → <identificatore> <identificatore abbreviato>
<prima classe> → <identificatore>
<seconda classe> → <identificatore>
<proprietà della associazione> → *attributi:* <attributo alfa> <lista attributi alfa> <vincoli>
<lista attributi alfa> → ε
<lista attributi alfa> → *;* <attributo alfa> <lista attributi alfa>
<attributo alfa> → <nome attributo> <cardinalità> *:* <dominio alfa>

Regole sui domini
<dominio> → *dominio* <identificatore dominio> *:* <definizione enumerato> .

Regole sugli strati topologici
<strato topologico> → *strato* <id strato> <proprietà strato>**.**
<proprietà strato> → *geometria:* <dominio strato> <vincoli>
<id strato> → <identificatore> <identificatore abbreviato>
<dominio strato> → *GU_Complex2D* | *GU_Complex3D* | *GU_CXCurve2D* |
GU_CXCurve3D | *GU_CXSurface2D* | *GU_CNCurve2D* |
GU_CNCurve2D | *GU_CXRing2D* | *GU_CXRing3D*

Regole sugli attributi a tratti strutturali e dinamici
Attributi a tratti strutturali
<classe tratto> → *tratto* <classe ausiliaria>
di <classe principale>.<attributo geo classe principale>
<proprietà classe ausiliaria>.
<classe ausiliaria> → <identificatore classe>
<classe principale> → <identificatore>
<attributo geo classe principale> → <identificatore>
<proprietà classe ausiliaria> → *attributi:* <attributo alfa> <lista attributi alfa>

Attributi a tratti dinamico
<classe evento lineare> → *eventoLineare*<classe ausiliaria>
di <classe principale>**.**<attributo geo classe principale>
unitàMisura <unità> *metodo* <metodo riferimento lineare>
<proprietà classe evento>**.**
<classe ausiliaria> → <identificatore classe>
<classe principale> → <identificatore>
<attributo geo classe principale> → <identificatore>
<unità>→ *centimetri* | *metri* | *chilometri* | *decametri* | *miglia*
<metodo riferimento lineare> → *relativo a* <classe dei riferimenti> | *assoluto*
<classe dei riferimenti> → <identificatore>
<proprietà classe evento> → *attributi:* <attributo alfa> <lista attributi alfa>
<classe evento puntiforme> → *eventoPuntiforme* <classe ausiliaria>
di <classe principale>**.**<attributo geo classe principale>
unitàMisura <unità> *metodo* <metodo riferimento lineare>
<proprietà classe evento>**.**

Regole sugli attributi a sottoaree strutturali

<classe sottoarea> → *sottoarea* <classe ausiliaria>
di <classe principale>.<attributo geo classe principale>
<proprietà classe ausiliaria>.

Regole sui vincoli topologici

<vincolo topologico> → *Vtopo* <classe vincolata> <selezioneX> <boundary>
<relazione topo> <lista relazioni> <tipo vincolo>
<classe vincolante> <selezioneYX> <boundary> <or lista vincoli>
<boundary> → ε | *.bnd* | *.pln* | *.bnd.pln* | *.pln.bnd*
<tipo_vincolo> → *esiste* | *union* | *perOgni*
<classe vincolata> → <identificatore>.<identificatore_attr_geo><B3D'>
<B3D'> → .<B3D> | ε
<B3D> → *S* | *B*
<classe vincolante> → <identificatore><id ruolo>.<identificatore_attr_geo><B3D'>
<identificatore_attr_geo> → <identificatore>
<id ruolo> → .<identificatore> <tipo ruolo> | ε
<tipo ruolo> → ::<identificatore> | ε
<lista relazioni> → ε
<lista relazioni> → , <relazione topo> <lista relazioni>
<relazione topo> → *DJ* | *TC* | *IN* | *CT* | *EQ* | *CR* | *OV*
<selezioneX> → ε | *(* <qualsiasi espressione boolean in OCL sugli attributi della classe vincolata X> *)*
<selezioneYX> → ε | *(* <qualsiasi espressione boolean in OCL sugli attributi della classe vincolante Y e eventualmente della classe vincolata X> *)*
<or lista vincoli> → *or* <vincolo topologico> <lista vincoli dis> | ε
<lista vincoli dis> → ε
<lista vincoli dis> → , <vincolo topologico> <lista vincoli dis>

Regole sui vincoli di struttura

<vincolo di struttura> → *struttura* <classe vincolata> <boundary str>
<tipo vincolo str> <classi vincolanti> <boundary str>
<classi vincolanti> → <classe vincolante> | (<classe vincolante> <lista classi vincolanti>)
<lista classi vincolanti> → , <classe vincolante> <lista classi vincolanti> | ε
<boundary str> → ε | *.bnd*
<tipo vincolo str> → *appartiene* | *compostoDa* | *appartiene* | *dj-appartiene* | *qdj-appartiene* | *partizionato* | *q-partizionato*

Regole sugli identificatori

<identificatore> → <carattere> <identificatore'>
<identificatore'> → <carattere> <identificatore'> | ε
<carattere> → a | ... | z | A | .... | Z | _ | <cifra>
<cifra> → 0 | ... | 9
<sep> → "spazi, tab e caratteri speciali per indicare fine riga"
<cifra diversa da zero> → 1 | ... | 9