DECRETO 10 novembre 2011
Regole tecniche per la definizione delle specifiche di contenuto dei database geotopografici.
(Gazzetta Ufficiale n. 48 del 27/02/2012 - Supplemento ordinario n. 37).

Allegato 2

Il modello GeoUML – Regole di interpretazione delle specifiche di contenuto per i DataBase Geotopografici

INDICE

Introduzione

Obiettivi

Questo documento definisce il *modello GeoUML*.

Il modello GeoUML viene utilizzato per definire la parte strutturata, detta *Schema Concettuale*, di una Specifica di Contenuto.

Una Specifica di Contenuto contiene infatti delle porzioni scritte in testo libero e delle porzioni scritte seguendo delle precise regole di strutturazione. La parte strutturata o Schema Concettuale delle Specifiche di Contenuto è la parte delle specifiche che si ottiene eliminando tutte le porzioni formulate in testo libero.

Il "*Catalogo dei Dati Territoriali*" è una particolare Specifica di Contenuto definita tramite il modello GeoUML e quindi la sua interpretazione si basa sulle regole descritte in questo documento

Conformità di un Data Product a una Specifica di Contenuto

Una Specifica di Contenuto costituisce una definizione dei contenuti che un *Data Product* deve possedere per essere conforme alla specifica stessa.
Il termine Data Product è utilizzato, in aderenza agli standard ISO 19100 per indicare una raccolta organizzata e coerente di informazioni territoriali. Un Data Product può essere ad esempio costituito da un insieme di file o da un database.
Per essere conforme a una Specifica di Contenuto un Data Product deve essere conforme sia alle parti scritte in testo libero, sia allo Schema Concettuale.
Dato che l'interpretazione del testo libero si basa sulla comprensione del linguaggio naturale, la conformità a tali parti deve essere interpretata direttamente, e non richiede definizioni aggiuntive.
La stesura di uno Schema Concettuale segue invece delle regole ben precise, che permettono di ottenere una definizione più accurata delle caratteristiche che il contenuto di un Data Product deve possedere per essere conforme alla specifica.
L'insieme di tali regole costituisce il modello GeoUML ed è descritto in questo documento.
L'uso di un modello formale come il GeoUML per specificare lo Schema Concettuale rende possibile la *verifica automatica della conformità di un Data Product allo Schema Concettuale*.
Si sottolinea che la conformità di un Data Product allo Schema Concettuale, chiamata qui conformità intrinseca, costituisce solamente una parte delle verifiche di conformità che un Data Product deve soddisfare; altri tipi di collaudo rimangono necessari per verificare che il Data Product rappresenti correttamente il Mondo Reale (conformità reale).
Il concetto più importante ai fini della valutazione della conformità di un Data Product a una Specifica di Contenuto è quello di *classe*: una classe definisce un insieme di oggetti che possiedono proprietà omogenee. Un oggetto appartenente a una classe è chiamato *istanza della classe*. L'insieme delle istanze (di una classe) presenti in un Data Product è detto *popolazione* (della classe).
Una Specifica definisce un insieme di classi e le proprietà di ogni classe. Per quanto concerne il popolamento delle classi, i due tipi di conformità citati sopra diventano:

1. conformità intrinseca: ogni istanza contenuta nel Data Product deve appartenere a una classe definita dalla specifica e deve possedere le proprietà definite dalla specifica per quella classe

2. conformità reale: per ogni oggetto O del mondo reale che appartiene a una classe presente nella specifica e tale che la sua componente spaziale superi la soglia di acquisizione determinata dall'accuratezza metrica prevista alla scala di rilievo deve esistere un'istanza corrispondente nel Data Product.

Si osservi che il popolamento di una classe dipende dalle componenti spaziali e dalla scala di rilievo, ma la conformità intrinseca è indipendente da tali aspetti, perché si applica solamente alle istanze rilevate.

Livello concettuale e livello fisico – Modello Implementativo

Una Specifica di Contenuto in generale e il suo Schema Concettuale in particolare definiscono il contenuto che un Data Product deve possedere a livello concettuale, cioè *in maniera indipendente dalla tecnologia utilizzata per materializzarlo*.
Dato uno Schema Concettuale è possibile definire un insieme di regole che permettono di materializzare un Data Product che rappresenta i contenuti richiesti dallo Schema Concettuale su una particolare struttura fisica. Tali regole costituiscono un *Modello Implementativo*.
La principale motivazione per la separazione dello Schema Concettuale dal Modello Implementativo è costituita dalla possibilità di definire diversi Modelli Implementativi e di utilizzarli per materializzare lo stesso Schema Concettuale. Alcuni esempi pratici dell'utilità di tale possibilità sono i seguenti:

- diversi soggetti possono realizzare Data Product conformi alla stessa Specifica di Contenuto utilizzando tecnologie e strutture dati diverse;
- lo stesso soggetto può utilizzare diversi Modelli Implementativi per materializzare in maniera diversa lo stesso contenuto informativo in diverse fasi operative (ad esempio, un Modello Implementativo per gestire la produzione, un diverso Modello Implementativo per gestire l'operatività del database);
- le specifiche di contenuto rimangono valide anche a fronte di cambiamenti nelle tecnologie.

La separazione tra livello concettuale e livello fisico richiede però, nell'analizzare la conformità di un Data Product a uno Schema Concettuale, di tener conto anche del Modello Implementativo utilizzato per materializzare il Data Product.
La definizione di Modelli Implementativi esula dagli obiettivi di questo documento.

Popolamento alle diverse scale e conformità di un Data Product al National Core (NC)

Una usuale specifica di contenuto definisce in un unico modo la struttura e le proprietà delle istanze che devono popolare un Data Product conforme alla specifica stessa. Diremo che una specifica di tale tipo è una *Specifica Omogenea*. In generale, una specifica è interpretata come omogenea ed è implicito che tutti i suoi costrutti siano popolati. In particolare questa è anche l'ipotesi alla base dell'interpretazione di un Application Schema negli standard ISO 19100.

In alcuni casi, tra i quali ricadono anche il *Catalogo dei dati territoriali* e il *National Core*, la situazione è più complessa, perché si vuole una specifica nella quale lo stesso elemento può avere proprietà diverse in base alla scala alla quale vengono rilevate le sue istanze. Diremo che una specifica di questo tipo è una Specifica di Contenuto *Differenziata*.

In una specifica differenziata è possibile definire un certo numero di "livelli di scala"

(cioè di raggruppamenti di scale considerate indifferenziate tra loro ai fini della specifica) e dichiarare per ogni elemento informativo della specifica se esso deve essere popolato a ciascuno di tali livelli. In questo modo una specifica differenziata contiene al suo interno diverse specifiche omogenee: la specifica costituita dall'insieme complessivo degli elementi informativi e, per ogni livello di scala LS, la specifica costituita dagli elementi che devono essere popolati a livello LS.

Ad esempio, nel Catalogo dei dati territoriali sono definiti due livelli di scala:

- NC1, che raggruppa le scale 1000 e 2000
- NC5, che raggruppa le scale 5000 e 10000

e il documento definisce implicitamente 3 specifiche omogenee:

1. L'insieme complessivo di tutti gli elementi che costituiscono il *Catalogo dei dati territoriali*, che è la specifica più completa e onnicomprensiva
2. Il *National Core* di livello NC1, che contiene tutti gli elementi che costituiscono la specifica da adottare alle scale 1000 e 2000
3. Il *National Core* di livello NC5, che contiene tutti gli elementi che costituiscono la specifica da adottare alle scale 5000 e 10000

E' importante osservare che le regole di conformità di un Data Product a una specifica differenziata sono più complesse delle regole di conformità di un Data Product a una specifica omogenea.

Vediamo due esempi con riferimento al National Core:

1) L'attributo Fondo della classe Area Veicolare (AC_VEI) ha nel NC il seguente popolamento alle scale:

"popolato a scala 1000-2000" e "non popolato a scala 5000-10000"

Questo significa che singole istanze della classe AC_VEI hanno una diversa struttura di attributi in base alla scala alla quale sono rilevate. Questa differenza impatta la conformità intrinseca di un Data Product al NC.

2) la classe delle Unità Volumetriche (UN_VOL) ha il seguente popolamento alle scale:

"popolata a scala 1000-2000" e "non popolata a scala 5000-10000"

Questa indicazione significa che le istanze di Unità Volumetriche dovranno essere rilevate solamente laddove la scala di acquisizione è 1000-2000, ma non dove è 5000-10000. Questa differenza impatta la conformità reale, ma non quella intrinseca. Se però esistesse il vincolo che le UN_VOL sono utilizzate per comporre geometricamente altri oggetti, ad esempio gli edifici, allora questo vincolo non potrebbe essere soddisfatto alla scala 5000-10000 e quindi anche la conformità intrinseca ne risulterebbe affetta.

Dato che le regole di conformità di un Data Product a una specifica differenziata sono più complesse delle regole di conformità di un Data Product a una specifica omogenea e dato che gli standard di riferimento si riferiscono sempre a specifiche omogenee, questo documento è organizzato nel modo seguente:

1. prima (dal capitolo 2 al capitolo 7) definisce tutti i costrutti che costituiscono una specifica omogenea, senza prendere in considerazione gli aspetti di popolamento differenziato alle scale; in tal modo questi aspetti possono essere rapportati più facilmente alle definizioni presenti negli standard di riferimento
2. poi, al capitolo 8, tratta tutti gli aspetti dipendenti dal popolamento differenziato alle diverse scale che caratterizzano una specifica differenziata come il NC.

È importante infine osservare che questa differenza di strutturazione della specifica in base alla scala si applica a priori e non altera le regole relative alla soglia di acquisizione di un oggetto; quindi la decisione se un oggetto O appartenente alla classe C deve essere rilevato a una certa scala S applica in sequenza le due regole seguenti:

1. se la classe C *non* è popolata alla scala S, O non viene rilevato, altrimenti si valuta la successiva regola
2. se le dimensioni di O superano la soglia di acquisizione determinata dall'accuratezza metrica prevista per la scala S, O viene rilevato, altrimenti O non viene rilevato.

Aspetti Generali del modello GeoUML

Componenti del modello GeoUML

Il modello GeoUML è composto da un insieme di costrutti che consentono di definire formalmente lo schema concettuale di una specifica. I costrutti sono suddivisi in due categorie:

- gli ***Elementi Informativi***, che costituiscono tutti i componenti utilizzabili per definire la struttura dei contenuti informativi della specifica, e
- i ***Vincoli di Integrità***, che si applicano agli elementi informativi e definiscono le proprietà che i dati, contenuti in un qualsiasi Data Product conforme alla specifica, dovranno soddisfare.

Il modello GeoUML è descritto progressivamente in questo documento seguendo il seguente ordine:

1. Modello GeoUML base per specifiche omogenee (capitoli da 3 a 6): questi capitoli descrivono i costrutti del modello facendo riferimento a una specifica omogenea;
2. Gestione delle superfici collassate (capitolo 7): questo capitolo descrive il modo in cui il modello GeoUML tratta il problema del collassamento delle superfici, cioè la rappresentazione di componenti spaziali concettualmente areali che si riducono a linee o punti per ragioni di dimensione;
3. Costrutti del modello GeoUML per supportare specifiche differenziate (capitolo 8).

Modalità di definizione del Modello GeoUML

In linea di principio il Modello GeoUML costituisce una specializzazione degli standard 19103, 19107, 19109 prodotti dal ISO TC 211, che a loro volta fanno riferimento agli standard *OMG-UML V1.3* (Unified Modeling Language) e OCL (Object Constraint Language) incluso nello stesso standard *OMG-UML V1.3.*

Pertanto la comprensione approfondita di questo documento richiede di conoscere tali standard, in particolare i costrutti dei “class diagram” del modello UML e il linguaggio OCL utilizzato in UML per esprimere vincoli di integrità.

La formalizzazione del GeoUML potrebbe essere quindi fatta integralmente fornendo le regole di traduzione di uno Schema GeoUML in un corrispondente schema UML realizzato secondo tali standard e, in effetti, l’insieme di tali regole è stato definito ai fini della produzione di un AS per la generazione, secondo lo standard ISO 19136, del formato GML di un Data Product.

In questo documento si è invece preferito, per ragioni di semplicità e di livello di astrazione, seguire il seguente metodo di formalizzazione più articolato:

1. Tutti i costrutti di base del GeoUML (cioè tutti i costrutti ad eccezione di quelli indicati ai punti seguenti) sono definiti formalmente fornendo la loro traduzione nei costrutti definiti dagli standard citati;
2. Il modello geometrico, cioè l’insieme dei tipi geometrici utilizzabili nello Schema Concettuale, è definito in maniera autonoma, seguendo le modalità utilizzate dallo standard SFM (Simple Feature Model), ISO 19125-1, per evitare di doverlo definire come specializzazione di ISO 19107, che è uno standard eccessivamente vasto rispetto al livello raggiunto dall’attuale tecnologia;
3. Gli attributi dipendenti dalle geometrie (attributi a tratti, a eventi e a sottoaree) sono definiti in maniera astratta introducendo metodi specifici per accedere ai loro valori e alle corrispondenti geometrie (tratti, eventi, sottoaree); ciò consente di lasciare indefinita e specificabile a livello di Modello Implementativo la

modalità di rappresentazione (che può essere strutturale, tramite segmenti di geometria, oppure basata sull'introduzione di un'ascissa curvilinea);
4. I Vincoli di Integrità sono specificati fornendo la loro traduzione nei costrutti definiti dagli standard citati, che nel caso dei vincoli implica un uso massiccio del linguaggio OCL.

Per quanto riguarda l'uso del linguaggio OCL si precisa che la funzione *oclisKindOf ()* è semplificata a *isKindOf()* e si richiama che la notazione O.f.g, dove O è un oggetto e f e g sono funzioni che restituiscono insiemi di valori, restituisce un unico insieme e non un insieme di insiemi.

Questo documento ha come scopo principale quello di fornire una definizione formale, secondo le modalità illustrate sopra, del GeoUML. Tuttavia, esso contiene anche un certo numero di indicazioni esemplificative che non possono però, in caso di dubbio, essere considerate sostitutive della definizione formale, che costituisce l'interpretazione di riferimento.
La definizione formale è resa necessaria dal ruolo che svolgono le specifiche; le ambiguità nell'interpretazione delle specifiche costituiscono infatti una sorgente di incertezza nella valutazione di conformità di un Data Product alla specifica stessa. In particolare, la definizione formale supporta efficacemente la realizzazione di strumenti di verifica automatica della conformità intrinseca di un Data Product a una specifica.

Sintassi del linguaggio GeoUML

La sintassi principale del linguaggio GeoUML è in forma testuale, a differenza di quella degli standard citati, che è grafica. La sintassi testuale si basa sull'impiego di parole chiave in una forma non estremamente sintetica al fine di migliorare la leggibilità della specifica. In questo documento le parole chiave sono indicate in *corsivo sottolineato*, ad esempio *classe*, ma il formato esatto di stampa può variare senza modificarne il significato.
Inoltre, come già detto, in una Specifica di Contenuto lo Schema Concettuale vero e proprio è mescolato a porzioni descrittive di testo libero che devono essere sempre ben riconoscibili e separabili dallo Schema Concettuale.
Anche se la sintassi testuale è quella principale, nel senso che uno Schema Concettuale deve essere completamente definito utilizzando tale sintassi, è possibile integrare tale schema con alcuni diagrammi in forma grafica. Valgono su questo aspetto le seguenti regole e limitazioni:

1. I diagrammi sono considerati non convenienti per rappresentare la struttura interna delle classi (attributi, domini, ecc...), ma costituiscono un utile complemento alla forma testuale nella rappresentazione dei legami che sussistono tra classi diverse, cioè associazioni tra classi, gerarchie di ereditarietà tra classi e vincoli di integrità tra classi;
2. La forma grafica del GeoUML è pertanto definita solamente per queste componenti: classi, associazioni, gerarchie e vincoli;
3. Nel caso in cui, per errore, vi sia un'inconsistenza tra i diagrammi e le definizioni date nella parte testuale, *prevale sempre la forma testuale*.

Elementi Informativi di base del GeoUML

Introduzione

Gli elementi informativi di base del linguaggio GeoUML sono, nell'ordine in cui verranno presentati, i costrutti di:

- classe
- attributo (non geometrico)
- cardinalità
- dominio enumerato
- dominio gerarchico
- associazione
- ereditarietà
- attributo geometrico
- attributo di attributo geometrico
- chiave primaria
- strato topologico

Tutti i costrutti di base del modello GeoUML possiedono le seguenti proprietà:

- **Nome applicativo** (obbligatorio): è la parola (o insieme di parole) che identifica il costrutto nel contesto applicativo a cui la specifica si riferisce.
- **Codice** (obbligatorio ad eccezione dei vincoli): è un codice univoco alfanumerico che identifica il costrutto.
- **Codice alfanumerico** o **abbreviazione** (obbligatorio per le classi, ma opzionale per gli altri costrutti): è un'abbreviazione del nome applicativo.

Tutti i costrutti di base del GeoUML sono definiti formalmente nel seguito fornendo la loro traduzione nei costrutti definiti dagli standard citati. In particolare, lo standard ISO 19109, *"Rules for Application Schema"* [19109] e gli standard collegati definiscono precisamente le regole per scrivere un Application Schema. Nel seguito si useranno indifferentemente i termini *AS* oppure *Schema UML-ISO* per indicare un Application Schema conforme a tali regole. Si richiama il fatto, importante per capire la logica del GeoUML, che un AS è costituito da un class diagram UML nel quale eventuali vincoli sono definiti utilizzando il linguaggio OCL.

Classe, Strato e Tema

Una classe possiede un nome applicativo, un codice e un codice alfanumerico (obbligatori e univoci nella specifica). La definizione testuale si basa sulla parola chiave *classe* seguita dal nome della classe e, tra parentesi, dal codice alfanumerico e dal codice.

Un oggetto appartenente a una classe è chiamato *istanza*; ogni oggetto è dotato implicitamente di un *identificatore* (OID) che non richiede di essere dichiarato a parte. L'insieme delle istanze presenti in un certo contesto (ad esempio, un Data Product) è detto *popolazione* (della classe).

Una classe può essere astratta. Una classe astratta non può avere istanze dirette, cioè le sue uniche istanze sono quelle appartenenti alle sue sottoclassi (il concetto di sottoclasse è definito più avanti) e pertanto può essere definita solo come superclasse di una gerarchia di classi al fine fattorizzare la rappresentazione di proprietà comuni a più sottoclassi. Sintatticamente una classe astratta si indica con la parola chiave *classe*

astratta invece di *classe*.

Le classi possono essere raggruppate per comodità in *Strati* e *Temi*, formando una gerarchia nella quale diverse classi appartengono a un unico Tema e diversi Temi appartengono a un unico Strato. E' importante sottolineare che la organizzazione delle classi in Strati e Temi è utile per la leggibilità della specifica, ma non ha nessuna rilevanza sul suo significato e quindi sulla conformità di un Data Product alla specifica stessa; cambiando tale strutturazione la specifica rimane inalterata e i Data Product che erano conformi rimangono conformi.
Nota Bene: la nozione di Strato, qui definita, non ha nessun legame con la nozione di "Strato Topologico" che verrà introdotta alla fine di questo capitolo. La condivisione del termine è solo la conseguenza dell'esigenza di mantenere continuità con terminologie utilizzate in precedenza. Il termine strato utilizzato qui potrebbe essere convenientemente sostituito con il termine "Categoria Tematica".

Trasformazione in AS.

Regola Classe Data una classe `C` si genera una classe `FeatureType` con lo stesso nome nello schema UML-ISO.

Esempio di applicazione della regola *Classe*

Costrutto GeoUML (forma testuale e forma grafica)	Traduzione in application schema UML-ISO
classe: C (abbreviazione - codice) ... C (abbreviazione - codice)	<<FeatureType>> C ...

Attributo (non geometrico) – domini di base

La nozione di ***attributo*** fa riferimento ad una proprietà di una classe esprimibile attraverso un valore scelto in un *dominio*.
I domini degli attributi non geometrici possono essere:

- domini di base (descritti in questa sezione)
- dominio enumerato (descritto più avanti)
- dominio enumerato gerarchico (descritto più avanti)
- dominio DataType (descritto più avanti)

I domini di base sono: *String, NumericString, Integer*, *Real*, *Boolean, Date, Time, DateTime*.

- *String* rappresenta una sequenza di caratteri di lunghezza finita,
- *NumericString* rappresenta una sequenza di cifre di lunghezza finita,

- *Integer* rappresenta i numeri interi,
- *Real* i numeri reali in virgola mobile,
- *Boolean* i valori di verità vero e falso,
- *Date* raccoglie i valori di tipo data nel formato: gg/mm/aaaa,
- *Time* i valori di tipo ora nel formato: hh:mm:ss,
- *DateTime* individua valori di timestamp formati da una data e un'ora nel formato: gg/mm/aaaa hh:mm:ss.

Ogni attributo ha un nome che deve essere univoco nell'ambito della classe (a parte le gerarchie), un codice univoco nella specifica, può avere un codice alfanumerico (opzionale) ed è sempre associato al nome del dominio (di base o altri) che caratterizza i valori ammissibili.

Per motivi legati all'implementazione e all'interoperabilità è necessario indicare, nei tipi *String* e *NumericString* un parametro che indica la lunghezza massima delle stringhe rappresentate. Quindi nella specifica tali tipi assumo la forma: String(N) e NumericString(N), dove N rappresenta la lunghezza massima.

La definizione testuale si basa sulla parola chiave *attributi* seguita dal codice, dal nome e dal tipo degli attributi; inoltre gli attributi non geometrici della classe sono distinti dagli altri attributi dalle parole chiave *attributi della classe*

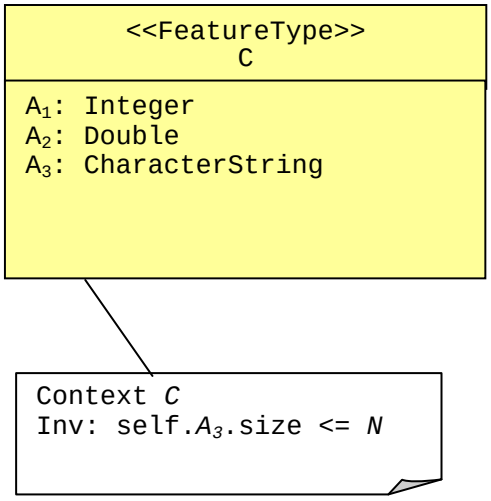
Trasformazione in AS.

I domini di base vengono trasformati nei corrispondenti domini del AS secondo la seguente tabella

Tipo GeoUML-base	Tipo in UML-ISO
Integer	Integer
Real	Double
String(N)	CharacterString
NumericString(N)	CharacterString con restrizione ai soli caratteri che rappresentano cifre.
Boolean	Boolean
Time	Time
Date	Date
DateTime	DateTime

Nel caso in cui in una classe C sia specificato un attributo A il cui dominio sia String(N) o NumericString(N) allora va aggiunto alla classe il seguente vincolo OCL:

```
Context C
Inv: self.A.size <= N
```

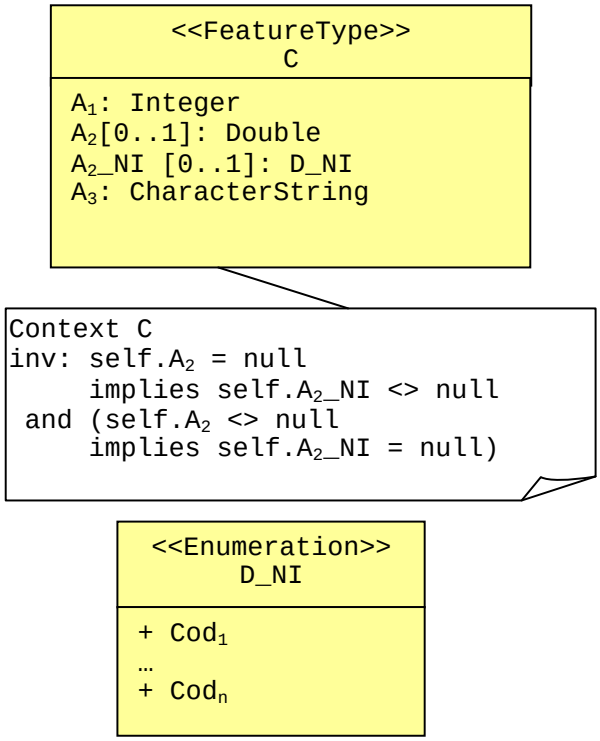
Costrutto GeoUML (forma testuale)	Traduzione in application schema UML-ISO
classe C (abbreviazione - codice) *attributi*: *attributi della classe*: A_1: *Integer* A_2: *Real* A_3: *String(N)*	

Cardinalità degli attributi (attributo multivalore)

Per rappresentare la cardinalità degli attributi nella notazione testuale si usa la stessa forma usata nella rappresentazione grafica (UML standard).

La cardinalità può essere applicata a tutti i tipi di attributi e i valori ammessi sono [0..1], [1..1], [0..*] e [1..*] e nel caso in cui sia omessa è assunto il valore di default [1..1]; la cardinalità "*" ammette che sia associato ad un attributo un insieme (senza duplicati) di valori.
L'opzionalità del valore dell'attributo (cardinalità minima uguale a zero) significa che è possibile assegnare a tale attributo il valore *nullo*; tale valore costituisce un valore speciale del GeoUML la cui implementazione in diversi Modelli Implementativi può variare.
Si noti che non sono ammessi la stringa vuota o l'attributo multivalore vuoto (insieme vuoto) come sostituzione del valore nullo negli attributi opzionali (e tantomeno negli attributi obbligatori).
Dato che il valore nullo può avere origini e significati diversi, è possibile associare ad ogni valore nullo un'etichetta presa dal dominio "*Null_Interpretation (D_NI)*". Tale dominio deve essere configurato per ogni specifica di contenuto (in assenza di configurazione tale dominio è vuoto).

Trasformazione in AS.
La cardinalità viene riportata nel AS esattamente come definita in GeoUML. Le etichette del valore nullo per un attributo A vengono rappresentate da un attributo aggiuntivo A_NI con dominio D_NI (dato che D_NI è un dominio enumerato, per capire bene la regola di trasformazione indicata è opportuno leggere il paragrafo successivo, relativo ai domini enumerati).

Costrutto GeoUML (forma testuale)	Traduzione in application schema UML-ISO
classe C (abbreviazione - codice) *attributi*: *attributi della classe*: A_1: *Integer* A_2 [0..1]: *Real* A_3: *String*	

Dominio enumerato

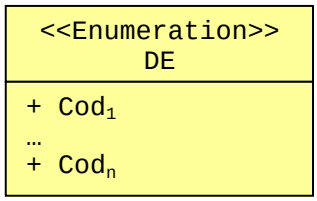
Un ***dominio enumerato*** è un dominio finito i cui valori sono predefiniti ed elencati nello schema.
La definizione di un attributo enumerato ha due forme: i valori possono essere elencati direttamente nella definizione dell'attributo, in tal caso si parla di "*dominio embedded*" e tale dominio è strettamente legato all'attributo e viene cancellato quando si cancella l'attributo, oppure essere elencati in una definizione separata del dominio.
Un dominio enumerato ha il codice, il nome e può avere il codice alfanumerico.

Dominio enumerato a definizione separata

Trasformazione in AS

Regola Dominio-enumerato
Dato un dominio enumerato `DE` si genera una classe `Enumeration DE` dove tutte le coppie (*codice*, *valore*) del dominio `DE` vengono rappresentate nella classe `Enumeration DE` tenendo il solo *codice* come istanze della classe `Enumeration DE`. Il *valore* viene aggiunto come tag value `documentation`.

Esempio di applicazione della regola *Dominio-enumerato*

Costrutto GeoUML (forma testuale)	Traduzione in application schema UML-ISO
dominio DE (abbreviazione - codice) Cod_1 Val_1 ... Cod_n Val_n	 tag values: Cod_1: documentation = Val_1 … Cod_n: documentation = Val_n

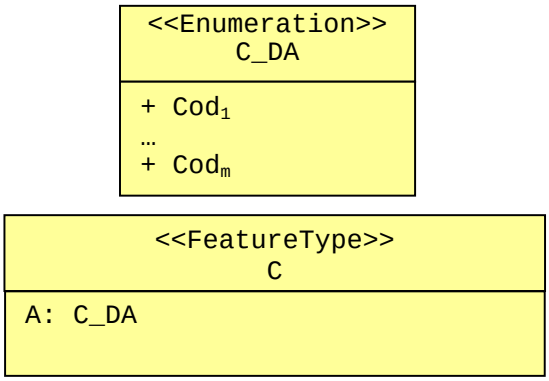
Dominio enumerato embedded

Questa forma più sintetica è indicata quando i valori sono pochi e il dominio non deve essere condiviso con altri attributi, altrimenti è preferibile la forma con definizione separata del dominio.

Trasformazione in AS

Regola Dominio enumerato embedded
Dato un dominio enumerato `DE embedded in una classe C` si genera una classe `Enumeration` 'ad hoc' di nome `C_DE` dove tutte le coppie (*codice*, *valore*) del dominio `DE` vengono rappresentate nella classe `Enumeration DE` tenendo il solo *codice* come istanze della classe `Enumeration DE`. Il *valore* viene aggiunto come tag value `documentation`.

Esempio di applicazione della regola Dominio enumerato embedded

Costrutto GeoUML (forma testuale)	Traduzione in application schema UML-ISO
classe C (abbreviazione - codice) *attributi*: *attributi della classe*: A: *Enum* *dominio (DA)* Cod_1 – Val_1 ..... Cod_m – Val_m	 tag values: Cod_1: documentation = Val_1 … Cod_m: documentation = Val_m

Dominio enumerato gerarchico

In alcuni casi è necessario rappresentare attributi i cui valori enumerati sono definiti attraverso una classificazione gerarchica.
Si ottiene questo tramite una versione arricchita dell'attributo enumerato, detta attributo enumerato gerarchico, nella quale dopo ogni valore della lista è possibile inserire un attributo di dominio enumerato (che rappresenta un livello aggiuntivo della gerarchia) e uno o più attributi aggiuntivi di tipo base; non può quindi essere aggiunto un attributo di dominio DataType, di tipo geometrico, o di un altro dominio enumerato gerarchico. Il nome dell'attributo di dominio enumerato può essere omesso.
Il dominio enumerato gerarchico ha come quello enumerato un nome, un codice e può avere un codice alfanumerico.

Si consideri il seguente esempio di attributo enumerato gerarchico con dominio embedded:

classe ElementoStradale (ELESTR – 0503)
attributi:
attributi della classe:
050301 – Tipo: *Enum*
dominio (DTipo)
01 – Piazza
0101 - rotatoria
0102 - non-rotatoria
02 – AreaStrutturata
sottotipo:
0201 - casello
0202 – incrocio
dimensione: real
03 – Tronco
lunghezza: real

l'attributo "tipo" sarebbe un semplice enumerato se fosse definito come:

050301 – Tipo: *Enum*
dominio (DTipo)
01 – Piazza
02 – AreaStrutturata
03 – Tronco

invece, dopo il primo valore "Piazza" troviamo un nuovo attributo di tipo enumerato (il cui nome è stato omesso), il cui dominio è (rotatoria, non-rotatoria) e rappresenta un nuovo livello della gerarchia, e dopo il secondo valore "AreaStrutturata" troviamo due attributi, "sottotipo" e "dimensione", dei quali uno è enumerato (rappresenta un ulteriore livello della gerarchia) e l'altro real, ecc...

Si noti che si può nidificare questo tipo di attributo anche più profondamente e che si può utilizzare la definizione separata del dominio enumerato invece dell'elencazione diretta dei valori utilizzata nell'esempio, secondo le regole viste nel paragrafo relativo agli attributi enumerati.

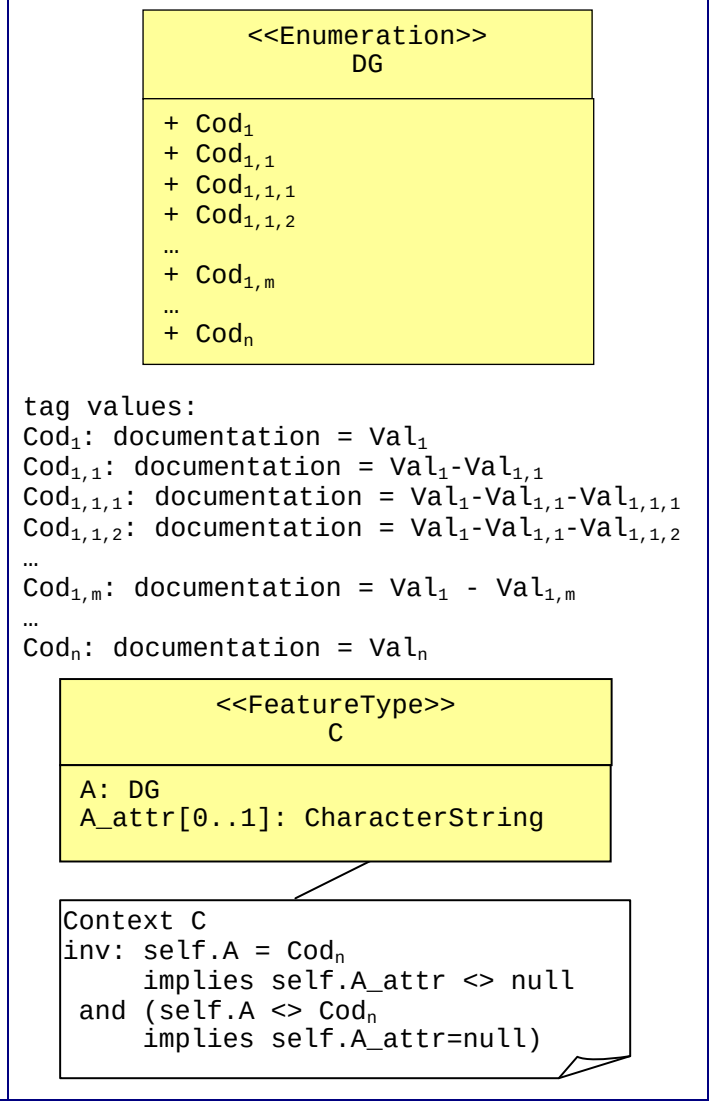
Trasformazione in AS

Regola Dominio-gerarchico

Questa regola è basata su due passaggi: il primo consiste nel trasformare il dominio gerarchico in un corrispondente dominio enumerato del GeoUML, il secondo consiste nel trasformare l'enumerato in AS secondo la regola già vista. La regola da applicare per il primo passaggio è la seguente:
Dato un dominio gerarchico `DG` si genera il corrispondente dominio enumerato `DG'` dove tutti i valori del dominio `DG` ai diversi livelli vengono generati come valori distinti del dominio enumerato `DG'`. Per ogni attributo normale `attr` eventualmente presente nel dominio gerarchico `DG` si genera, nella classe `C` contenente l'attributo `A` di dominio `DG`, un attributo normale opzionale `A_attr` e si aggiunge un vincolo OCL per indicare che tale attributo deve essere valorizzato solo nel caso in cui l'attributo `A` assuma un certo insieme di valori.

Di seguito si riporta un esempio di applicazione della regola.

Esempio di applicazione della regola *Dominio-gerarchico*

Costrutto GeoUML (forma testuale)	Traduzione in application schema UML-ISO
dominio DA (abbreviazione - codice) $Cod_1 – Val_1$ $Cod_{1,1} – Val_{1,1}$ $Cod_{1,1,1} – Val_{1,1,1}$ $Cod_{1,1,2} – Val_{1,1,2}$ … $Cod_{1,m} – Val_{1,m}$ $Cod_n – Val_n$ attr: *String* *classe* C (abbreviazione - codice) *attributi*: *attributi della classe*: A: *Enum* DA	

Il dominio DataType

Il dominio *DataType* consente di arricchire l'insieme dei domini disponibili per gli attributi con un dominio strutturato, i cui valori sono record di valori. I componenti

(attributi) di tale record possono contenere valori appartenenti ai domini di base, enumerati e enumerati gerarchici, ma non ai DataType e ai tipi geometrici.
Gli attributi del DataType possono avere una cardinalità 1..1 oppure 0..1. Poiché un attributo associato al dominio DataType ha a sua volta una cardinalità se ne evidenziano le implicazioni:

- ogni record di valori deve sempre rispettare la cardinalità definita dai propri componenti;
- nel caso in cui l'attributo con dominio DataType abbia cardinalità minima 1 deve esistere almeno un record e in ogni record deve esistere almeno un componente con valore diverso da nullo, indipendentemente dalla cardinalità dei componenti.

I DataType si differenziano dalle classi per il fatto di non avere un'identità, essi possono essere utilizzati solo come dominio (tipo) di un attributo.
La definizione testuale si basa sulla parola chiave *DataType* seguita dal nome (con codice univoco nella specifica e codice alfanumerico opzionale) e dalla lista degli attributi.

Trasformazione in AS.

Regola DataType
Dato un DataType `TD` si genera una classe `DataType` di UML-ISO. Gli attributi del DataType vengono rappresentati come attributi della classe `DataType` rispettando le cardinalità. I domini degli attributi vengono tradotti come indicato precedentemente; se è presente un dominio enumerato embedded `DE`, nella regola vista precedentemente la classe `Enumeration` 'ad hoc' che viene generata per rappresentare i suoi valori ha il nome `TD_DE` (cioè assume in questo caso il nome del tipo di dato invece di quello della classe).

Esempio di applicazione della regola *Tipo-di-Dato*

Costrutto GeoUML (forma testuale)	Traduzione in application schema UML-ISO
DataType TD (abbreviazioni) *attributi*: A_1: *Integer* A_2 [0..1]: *Real* A_3: *String*	<<DataType>> TD A_1: Integer A_2[0..1]: Double A_2_NI[0..1]: D_NI A_3: CharacterString

Associazione (binaria) senza attributi

La notazione testuale per la definizione di associazioni consiste nel dichiarare, dopo la parola chiave *ruoli*, il ruolo della classe collegata (indicando il codice univoco, il codice alfanumerico e il nome) ed eventualmente, dopo la parola chiave *inverso*, il ruolo opposto.

Trasformazione in AS.

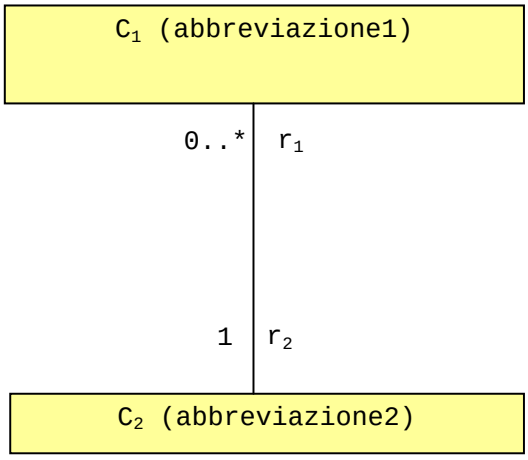
Regola Associazione-binaria-senza-attributi Data un'associazione A tra le classi C_1 e C_2 con i rispettivi ruoli r_1 e r_2 si genera un'associazione con gli stessi ruoli tra le corrispondenti classi `FeatureType` dello schema UML-ISO.

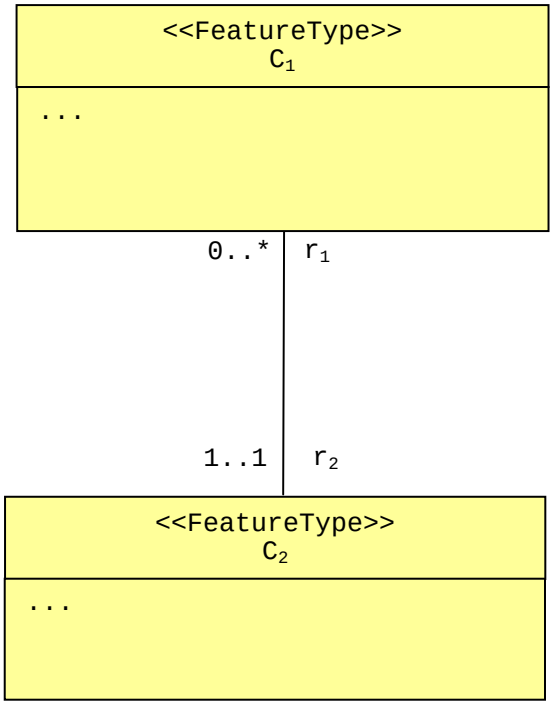
Esempio di applicazione della regola *Associazione-binaria-senza-attributi*

Costrutto GeoUML (forma testuale e forma grafica)	**Traduzione in application schema UML-ISO**

classe C1 (abbreviazione1 – codice1)
attributi:
ruoli:
r_2: [1]C2
inverso r_1 [0..*]

classe C2 (abbreviazione2 – codice2)
attributi:
ruoli:
r_1 [0..*]: C1
inverso r_2 [1]





Si noti che nella forma testuale è presente ridondanza, perché in questo esempio l'associazione e i ruoli sono stati definiti in ambedue le classi interessate, in forma simmetrica; questa ridondanza può essere evitata rinunciando alla dichiarazione dell'associazione in una delle due classi, a fronte di una maggiore difficoltà per il lettore di capire in quali associazioni è coinvolta una classe.

Associazione (binaria) con attributi

Talvolta un'associazione possiede attributi propri. In questo caso la forma testuale richiede di dichiarare, in maniera indipendente rispetto alle dichiarazioni dei ruoli all'interno delle classi interessate, anche un'associazione, utilizzando la parola chiave *associazione*. I ruoli coinvolti nell'associazione devono avere entrambi cardinalità

massima “*”.

Trasformazione in AS.

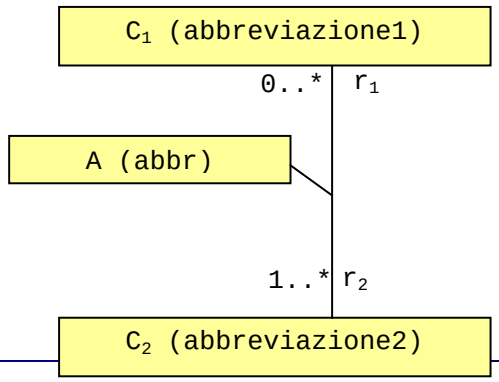
Regola Associazione-binaria-con-attributi
Se l’associazione A tra le classi C_1 e C_2 presenta degli attributi, si genera un FeatureType A dello schema UML-ISO che corrisponde all’associazione A e include tutti gli attributi di A. Tale FeatureType A viene collegato alle due classi coinvolte nella relazione attraverso due associazioni binarie che conservano i ruoli r_1 e r_2 verso le classi C_1 e C_2, mentre si introducono i ruoli A_r_1 e A_r_2 verso il Feature Type A.

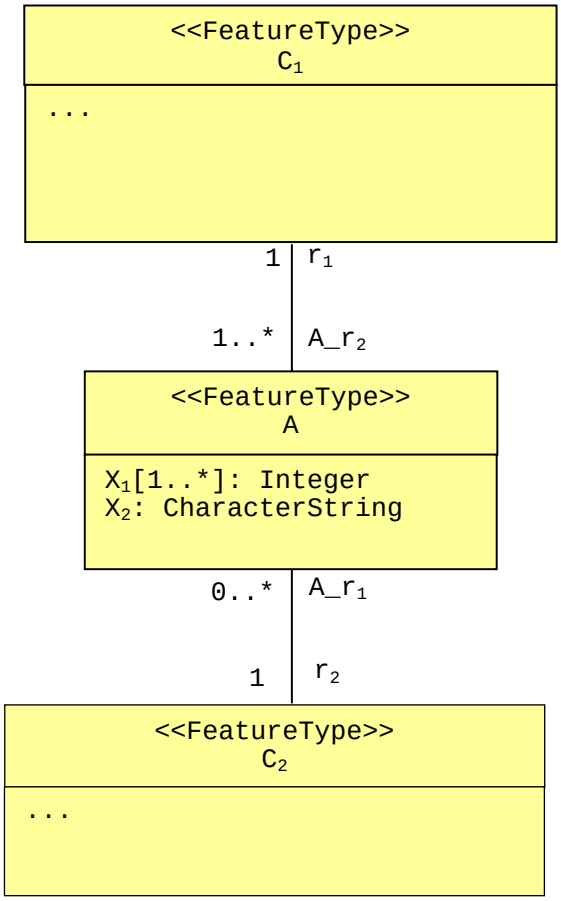
Esempio di applicazione della regola *Associazione-binaria-con-attributi*

Costrutto GeoUML (forma testuale e forma grafica)	Traduzione in application schema UML-ISO

classe C1 (abbreviazione1 – codice1)
attributi: …
ruoli:
associazioni:
A (abbr)
attributi dell’associazione:
X_1 [0..*]: *Integer*
X_2: *String*
ruoli dell’associazione:
r_2 [1..*]: C2 *inverso* r_1
[0..*]

classe C2 (abbreviazione2 – codice2)
attributi: …
ruoli:
associazioni:
A (abbr)
attributi dell’associazione:
X_1 [0..*]: *Integer*
X_2: *String*
ruoli dell’associazione:
r_1 [0..*]: C1 *inverso* r_2
[1..*]





Ereditarietà tra classi

L'ereditarietà tra classi si definisce dichiarando la relazione di sottotipo tra una classe (detta sottoclasse) e un'altra classe più generale (detta superclasse). Un esempio di definizione in GeoUML di una gerarchia di ereditarietà semplice in notazione testuale è il seguente:

classe AreaCircolazione (ACIR – 0203)
superclasse *incomplete* (AreaCircolazioneStradale)
attributi:
...;
classe AreaCircolazioneStradale (ACST - 0204)
sottoclasse di AreaCircolazione
...

Questa dichiarazione implica che:

- tutti gli oggetti della sottoclasse appartengano alla superclasse, ma non viceversa (nell'esempio si dichiara quindi che possono esistere *Aree di Circolazione* che non sono *Aree di Circolazione Stradale*) e la gerarchia viene chiamata **incompleta** (aspetto estensionale);
- dalla precedente proprietà si deriva che tutte le proprietà della superclasse si applicano agli oggetti della sottoclasse, ossia la sottoclasse eredita gli attributi, i ruoli (associazioni), le loro cardinalità e i vincoli nei quali la superclasse è vincolata (aspetto intensionale);
- la sottoclasse può aggiungere alle proprietà ereditate propri attributi, ruoli e vincoli; in questo caso il nome assegnato ai propri attributi e ruoli deve essere univoco nella classe considerando tuttavia anche le proprietà ereditate. I vincoli aggiunti possono utilizzare sia gli attributi/ruoli propri della classe che quelli ereditati.

Una gerarchia di ereditarietà in generale può coinvolgere più di una sottoclasse; in tal caso la forma testuale è la seguente:

classe AreaCircolazione (ACIR – 0203)
superclasse (AreaCircolazioneStradale,
AreaCircolazionePedonale, AreaCircolazioneCiclabile)
attributi:
...;

Si precisa che con questa forma in generale si vuole definire una gerarchia **completa** nella quale ogni oggetto della superclasse appartiene almeno ad una sottoclasse (ovviamente continua a valere anche la proprietà che ogni oggetto di una sottoclasse appartiene alla superclasse). Inoltre la gerarchia può essere **disgiunta** se un oggetto della superclasse non può appartenere contemporaneamente a più sottoclassi, altrimenti la gerarchia è **sovrapposta**.
Le proprietà di completezza e disgiunzione sono definite dopo la parola chiave *superclasse* utilizzando le parole chiave dello standard UML: *incomplete* (se omessa la gerarchia è considerata *complete) e overlapping* (se omessa la gerarchia è considerata *disjoint*). Nel seguente esempio si definisce in forma testuale una gerarchia incompleta e disgiunta:

classe AreaCircolazione (ACIR – 0203)
superclasse *incomplete* (AreaCircolazioneStradale,
AreaCircolazioneCiclabile)

attributi: ...

Con questa dichiarazione possono esistere *Aree di Circolazione* che non sono né *Aree di Circolazione Stradale* né *Aree di Circolazione Ciclabile*, ma non possono esistere *Aree di Circolazione* che siano sia *Aree di Circolazione Stradale* e sia *Aree di Circolazione Ciclabile*, perché non abbiamo eliminato la disgiunzione (per ottenere quest'ultimo effetto avremmo dovuto scrivere *superclasse* *incomplete* *overlapping*).

Poiché una superclasse può essere a sua volta sottoclasse di un'altra classe è possibile generare una gerarchia di ereditarietà a più livelli. In tal caso un oggetto di una classe appartiene anche a tutte le classi antenate dirette o indirette nella gerarchia e la classe eredita le proprietà di tutte le classi antenate della gerarchia. Ciò significa che un'associazione o un vincolo tra due classi coinvolgerà gli oggetti delle due classi e gli oggetti di tutte le classi definite nella porzione di gerarchia di ereditarietà della quale le due classi specificate ne rappresentano la radice.

Commento

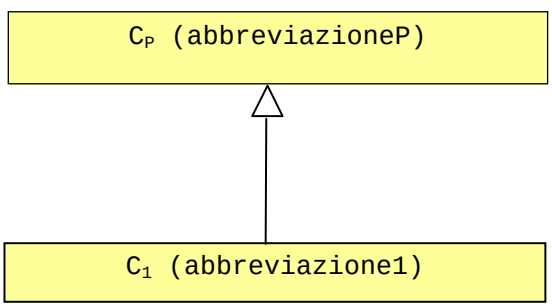
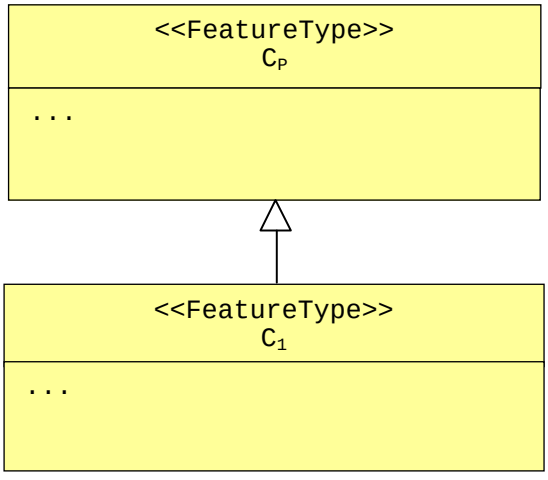
Una gerarchia di ereditarietà coinvolge spesso classi astratte al fine di definire proprietà che le classi concrete metteranno poi a disposizione; per questo motivo esse sono spesso collocate alla radice della gerarchia. Una classe astratta è sempre superclasse di una gerarchia "complete" in quanto essa non può avere istanze proprie.

Trasformazione in AS.

Regola Gerarchia
Data una gerarchia G tra una classe padre C_p e una o più classi figlie C_1 ,…, C_n dello schema GeoUML si genera una gerarchia equivalente tra le corrispondenti classi `FeatureType` dello schema UML-ISO.

Di seguito si riporta un esempio di applicazione della regola sopra esposta.

Esempio di applicazione della regola *Gerarchia*

Costrutto GeoUML (forma testuale e forma grafica)	Traduzione in application schema UML-ISO
classe C_p (abbreviazioneP - codiceP) *superclasse* (C_1) *attributi*: *classe* C_1 (abbreviazione1 – codice1) 	

Attributo geometrico (o componente spaziale)

Un attributo geometrico, detto anche componente spaziale, è un attributo i cui valori appartengono ad uno dei tipi geometrici che costituiscono i domini di tale attributo. La descrizione di tali domini è fornita nel Modello Geometrico, descritto nel prossimo capitolo.
La cardinalità di un attributo geometrico può essere solamente [1..1] o [0..1] (la molteplicità è trattata tramite il concetto di aggregato del modello geometrico).
L'assenza di geometria (cardinalità minima 0) è gestita attraverso il concetto di valore nullo, pertanto non è ammesso un attributo che memorizzi una geometria vuota.
Gli attributi geometrici sono distinti dagli altri attributi dalla parola chiave *componenti spaziali della classe*.
Una classe può possedere più di un attributo geometrico.

Attributo di attributo geometrico

Gli attributi geometrici possono essere ulteriormente descritti da attributi che precisano alcune caratteristiche della geometria che viene rappresentata nell'attributo geometrico.
Un attributo di attributo geometrico può essere definito su qualsiasi tipo di attributo geometrico, ha un nome univoco nell'ambito degli attributi della componente spaziale sulla quale è definito, ha un codice, un codice alfanumerico opzionale e può avere la cardinalità come gli attributi normali; infine, il suo dominio può essere qualsiasi dominio applicabile agli attributi normali della classe.

Costrutto GeoUML (forma testuale)	Traduzione in application schema UML-ISO
classe C *attributi* *attributi della classe* … *componenti spaziali della classe* g: GU_Object*D; *attributi di questa componente spaziale* A: D_A *su* g;	<<FeatureType>> C g: GU_Object*D g_A: D_A Context C inv: self.g = null implies self.g_A=null

Chiave primaria

Un identificatore di un oggetto è caratterizzato da due aspetti:

- il contesto di unicità (scope) che stabilisce quale sia l'ambito nel quale l'identificatore è univoco;
- la visibilità esterna che stabilisce se l'identificatore svolge anche il ruolo di collegamento con dati che sono all'esterno dello "scope".

In GeoUML si assume che tutti gli oggetti siano dotati di un identificatore automatico interno, ossia non visibile all'esterno, chiamato OID (object identifier) che li identifica in tutto il Data Product.
Oltre a questo identificatore il GeoUML ammette che siano definiti ulteriori identificatori, chiamati, come nella terminologia SQL, ***chiave primaria***, al fine di soddisfare la caratteristica di

visibilità esterna; in particolare permette di definire identificatori per la specifica classe applicativa che permettano di mantenere in modo persistente il legame con dati esterni. Si noti che a questi fini non può essere utilizzato un identificatore interno che un sistema può modificare se necessario (si pensi a certi identificatori di primitive geometriche in alcuni tipi di GIS).
Una chiave primaria è un identificatore definito all'interno di una classe ed è costituita da un insieme di attributi e/o ruoli di associazioni (senza attributi) che ha le seguenti proprietà:
- il dominio degli attributi deve essere uno di quelli di base, un dominio enumerato o un dominio gerarchico, ma senza attributi aggiuntivi;
- la cardinalità degli attributi e/o ruoli è [1..1].

Nel caso delle gerarchie di ereditarietà se una classe C definisce il vincolo di chiave primaria esso è ereditato da tutte le classi della sottogerarchia di cui C è la radice. In tal caso nessuna delle classi appartenenti a tale sottogerarchia può definire un ulteriore vincolo di chiave primaria. La definizione di una chiave primaria può utilizzare sia gli attributi/ruoli ereditati e sia quelli propri aggiuntivi.

Commento

La scelta di attribuire o meno una chiave primaria ad una classe deve essere valutata con attenzione perché può avere rilevanti conseguenze nel modello implementativo. Inoltre si noti che è possibile, in un modello implementativo, sostituire il concetto di OID con il concetto di *UUID* (Universal Unique Identifier), cioè con un identificatore dotato di visibilità esterna.

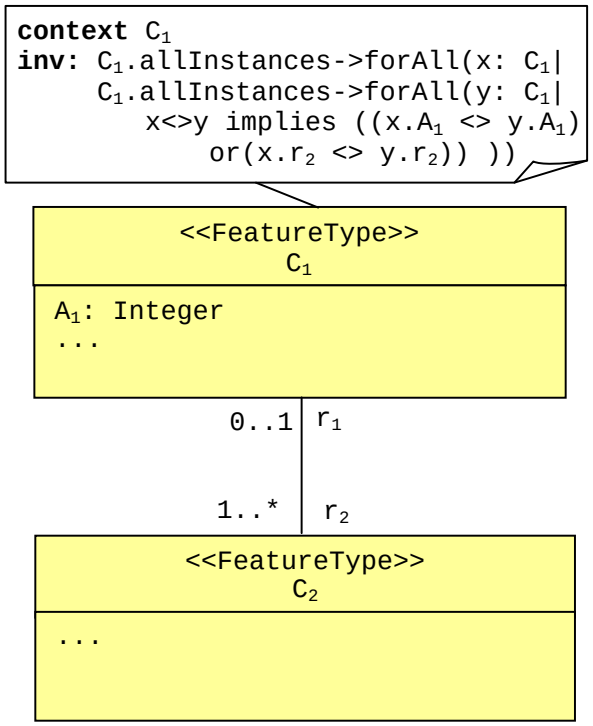
Trasformazione in AS

Regola Vincolo-chiave-primaria
Data una classe C con vincolo di chiave primaria sugli attributi $a_1, \ldots, a_n$ e sui ruoli $r_1, \ldots, r_m$, tale vincolo si rappresenta attraverso il seguente vincolo OCL aggiunto al Feature Type ottenuto per trasformazione della classe C:

```
context C
inv: C.allInstances -> forAll (x: C |
     C.allInstances -> forAll (y: C | x<>y implies
                 ((x.a1 <> y.a1) or … or (x.an <> y.an) or
                  (x.r1 <> y.r1) or … or (x.rm <> y.rm)) ))
```

Di seguito si riporta un esempio di applicazione della regola sopra esposta.

Esempio di applicazione della regola *Vincolo-chiave-primaria*

Costrutto GeoUML (forma testuale)	Traduzione in application schema UML-ISO
classe C_1 (abbreviazioni) *attributi*: *attributi della classe*: *PK* A_1: *Integer*; *ruoli*: *PK* r_2 [1..1]: C_2 ... *classe* C_2 (abbreviazioni) *attributi*: *attributi della classe*: .... *ruoli*: r_1 [0..1]: C_1 ...	

Strato topologico

Uno strato topologico è una classe dotata di due caratteristiche particolari:

- nei dati può esistere un solo oggetto di tale classe (classe mono-istanza)
- ha un unico attributo geometrico monovalore di nome ***geometria***.

Gli strati topologici vengono spesso utilizzati per imporre condizioni più stringenti sulla strutturazione di un gruppo di oggetti geometrici, ad esempio la definizione di coperture del suolo.

Trasformazione in AS.

Regola Strati-topologici
Dato uno strato topologico ST, esso si rappresenta come classe ST UML-ISO conservando gli attributi dello strato a cui si aggiunge il vincolo OCL:

```
context C
inv: C.allInstances->size <= 1
```

Di seguito si riporta un esempio di applicazione della regola sopra esposta.

Esempio di applicazione della regola *Strati-topologici*

Costrutto GeoUML (forma testuale e forma grafica)	Traduzione in application schema UML-ISO
Strato ST (abbreviazione) *Tipo geometrico*: GU_CXSurface2D <<Strato Topologico>> ST geometria: GU_CXSurface2D	ST geometria: GU_CXSurface2D **context** ST **inv:** ST.allInstances->size <= 1

Il Modello Geometrico di GeoUML

Caratteristiche generali degli oggetti e dei tipi geometrici

Il modello geometrico definisce un insieme di tipi che descrivono le possibili geometrie degli attributi geometrici.
I tipi geometrici permettono fondamentalmente la definizione di due categorie di oggetti geometrici:

- le **geometrie primitive:** valori geometrici atomici non ulteriormente divisibili composti da un singolo, connesso ed omogeneo elemento dello spazio (ad esempio, una curva);
- le **collezioni geometriche:** insiemi di geometrie elementari che possono essere omogenee nel tipo dei componenti (multi-punti, multi-curve o multi-superfici) o eterogenee e che possono imporre dei vincoli spaziali sui componenti.

I tipi geometrici sono correlati tra di loro e pertanto sono descritti attraverso la gerarchia dei tipi geometrici mostrata in Figura 4.1, utilizzando un diagramma delle classi UML; tale gerarchia lega i tipi tra di loro tramite la relazione di "sottotipo". Se un tipo è sottotipo di un altro, automaticamente ne eredita tutte le proprietà e poiché la relazione è transitiva un tipo eredita in generale le proprietà di tutti i tipi che sono suoi antenati nella gerarchia. Un tipo può poi arricchire le proprietà ereditate con ulteriori specificazioni oppure può porre dei vincoli restrittivi alle proprietà ereditate; le caratteristiche del tipo così definito sono poi a loro volte ereditate da tutti i suoi sottotipi.
La gerarchia permette di descrivere in modo incrementale le proprietà dei tipi geometrici a partire dalle proprietà comuni a tutti i tipi geometrici, descritte nel tipo *GU_Object*, radice della gerarchia. Si noti che nel diagramma di Figura 4.1 i tipi col nome in *corsivo* hanno lo stereotipo *abstract* (ad esempio, *GU_Object*) e rappresentano tipi astratti, ossia tipi che sono stati definiti allo scopo di descrivere proprietà generali dei tipi geometrici che devono poi essere rese disponibili dai sottotipi del tipo astratto; questi tipi non possono quindi essere utilizzati come tipi concreti da associare agli attributi geometrici del GeoUML.
Tutti gli oggetti geometrici del GeoUML sono definiti in un sistema di riferimento di coordinate. Il GeoUML distingue i tipi geometrici in:

- tipi che modellano gli oggetti geometrici senza coordinata Z e che per semplicità saranno chiamati tipi definiti nello spazio 2D e dei quali il tipo *GU_Object2D* è la radice;
- tipi definiti per gli oggetti dello spazio tridimensionale (chiamato nel seguito spazio 3D) con radice nel tipo *GU_Object3D*.

Si noti che le proprietà di un tipo, i vincoli che impone o le relazioni topologiche che lo coinvolgono considerano sempre tutte le coordinate previste dallo spazio nel quale il tipo è definito.
Lo schema mette inoltre in evidenza la relazione esistente tra i tipi aggregati e i tipi degli oggetti componenti.
Questo capitolo descrive matematicamente i valori di ogni tipo geometrico attraverso il concetto astratto di "insieme di punti" sul quale vengono poi definite le proprietà e/o i vincoli che devono essere soddisfatti.

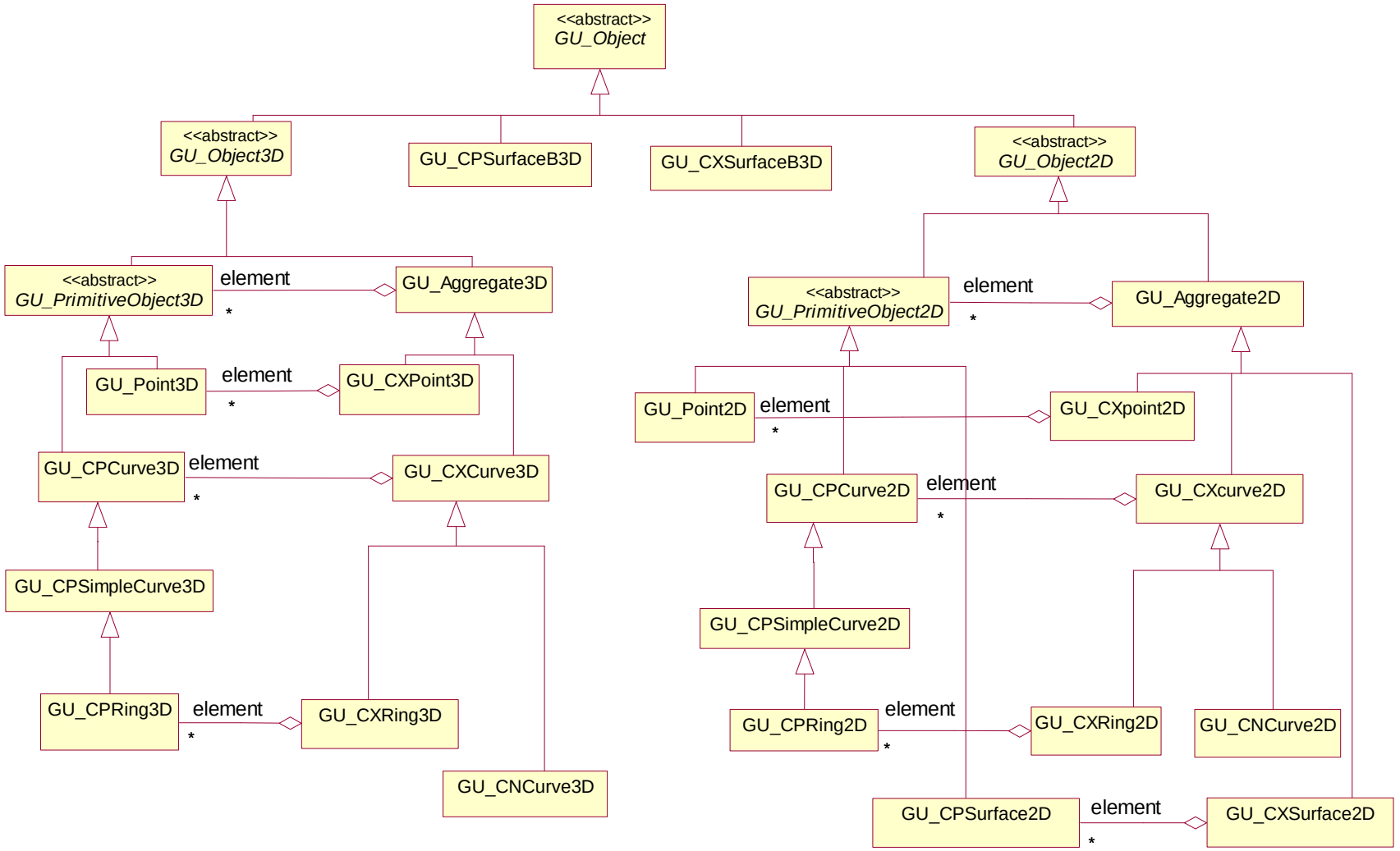


Figura 4.1 Gerarchia UML delle classi che rappresentano i tipi geometrici di GeoUML.

Si noti che rappresentazioni specifiche dei tipi geometrici, quale quella vettoriale, e i metodi di interpolazione, non sono considerate dal GeoUML, ma dai modelli implementativi che definiscono la rappresentazione utilizzata per la geometria nel Data Product associato alla specifica. In un certo senso quindi tutti i tipi utilizzati a livello concettuale sono astratti rispetto al livello fisico; ad esempio, il tipo GU_CPCurve a livello concettuale definisce le proprietà di una curva nello spazio euclideo e costituisce un tipo assegnabile a un attributo dello schema concettuale, ma la sua implementazione in un modello implementativo basato sul SFM dovrebbe essere di tipo Linestring, cioè basata sul concatenamento di segmenti con interpolazione lineare.
I diversi tipi geometrici sono descritti in Sezione 4.2, mentre gli operatori che verificano le relazioni topologiche tra gli oggetti dei tipi geometrici sono descritti in Sezione 4.3.

Commento

Il Modello Geometrico del GeoUML arricchisce il modello dello standard SFM principalmente nei seguenti modi:

- L'estensione allo spazio 3D delle geometrie puntiformi e lineari e delle loro relazioni topologiche
- L'estensione allo spazio 3D della frontiera delle superfici bidimensionali tramite il concetto di "superficieB3D"
- L'introduzione di alcune specializzazioni delle geometrie lineari

I nomi utilizzati per i tipi geometrici del GeoUML hanno una motivazione storica, legata a precedenti definizioni del modello geometrico, basate sui complessi dello standard 19107. In particolare, l'acronimo CP in CPCurve o CPSurface si riferiva al termine "composite" del 19107, ma non ha più alcun significato in questa definizione, nella quale CPCurve e CPSurface rappresentano geometrie primitive, e l'acronimo CX in CXCurve e in CXSurface assume un significato molto simile (ma non coincidente) a quello di "Multi" nello standard SFM.

Nel seguito, per semplificare il riferimento a più tipi geometrici contemporaneamente si utilizza il carattere * ; ad esempio C*curve*D sta per CPcurve2D oppure CPcurve3D oppure CXcurve2D oppure CXcurve3D.

I tipi geometrici del GeoUML

In questa sezione vengono descritti nel dettaglio i tipi geometrici del GeoUML precisandone le proprietà specifiche e definendo la semantica delle proprietà generali. Inoltre per ogni tipo viene assegnata la definizione matematica dei valori possibili.

Il tipo GU_Object

Definizione dei valori possibili

Il tipo *GU_Object* è un tipo astratto e definisce le proprietà generali di tutti gli oggetti geometrici definiti nel modello GeoUML.
Dal punto di vista matematico un oggetto di tipo *GU_Object* è un insieme infinito di punti (ad eccezione dei tipi che descrivono punti isolati):

1. definito in uno **spazio euclideo** $\Re^n$ nel quale le n coordinate della posizione del singolo punto sono assegnate in base al sistema di coordinate utilizzato. Il GeoUML definisce $\Re^2$ per gli oggetti definiti nello spazio 2D e $\Re^3$ per quelli definiti nello spazio 3D, dove la coordinata Z è tipicamente usata per rappresentare l'altitudine;
2. **topologicamente chiuso**, ossia l'insieme di punti che l'oggetto rappresenta include anche i punti che costituiscono la frontiera dell'insieme;
3. **regolare,** ossia l'unione della parte interna dell'insieme di punti e della sua frontiera coincide con l'insieme stesso; quest'ultima proprietà impedisce oggetti anomali

come ad esempio, poligoni che abbiano dei tagli nella parte interna o buchi composti da un solo punto.

Proprietà generali.

- **boundary():** *GU_Object*
  Restituisce la frontiera dell'oggetto geometrico che è costituita dall'insieme di punti che limitano l'estensione dell'oggetto, ossia quei punti che sono caratterizzati dalla seguente proprietà: preso un loro qualsiasi intorno, esso interseca sia la parte interna e sia la parte esterna dell'oggetto. La determinazione della frontiera avviene ipotizzando di inserire l'oggetto in uno spazio di coordinate che abbia la stessa dimensione dell'oggetto (ad esempio, uno spazio bidimensionale per le superfici e uno monodimensionale per le curve); questo vincolo permette di definire frontiere intuitive per gli oggetti (ad esempio, una curva in uno spazio monodimensionale ha come frontiera i suoi punti estremi, mentre in uno spazio di dimensione maggiore tutta la curva sarebbe frontiera di se stessa). La frontiera sarà quindi definita da oggetti che hanno sempre una dimensione inferiore a quella dell'oggetto considerato (ad esempio, la frontiera di una curva è costituita da punti). La definizione dettagliata dell'oggetto geometrico che rappresenta una frontiera richiede di considerare lo specifico sottotipo e verrà quindi descritta in seguito.
- **coordinateDimension():** Integer
  Restituisce la dimensione delle coordinate di un oggetto, ossia il numero di assi necessari per descrivere la posizione di ogni punto dell'oggetto geometrico in un sistema di coordinate. Tale valore dipende esclusivamente dal tipo al quale appartiene il singolo oggetto.
- **dimension()**: Integer
  Restituisce la dimensione intrinseca dell'insieme di punti associato all'oggetto geometrico; essa dipende dal tipo dell'oggetto considerato (ad esempio, 1 per una curva) ad eccezione dell'aggregato generico ed ha la proprietà di essere sempre inferiore o uguale alla dimensione delle coordinate dell'oggetto.
- **isCycle()**: Boolean
  Ritorna TRUE se l'oggetto geometrico è ciclico (il termine ciclico è spesso sostituito dal concetto di "chiuso su se stesso" quando non esiste la possibilità di confondere quest'ultimo termine col concetto di topologicamente chiuso). Se ne deriva che un oggetto ciclico non abbia frontiera.
- **isSimple()**: Boolean
  Ritorna TRUE se l'oggetto geometrico è semplice, ossia se non possiede punti di autointersezione o autotangenza.
- **spatialReferenceSystem()**: Integer
  Restituisce l'identificatore internazionale del sistema di riferimento di coordinate dell'oggetto.
- **planar()**: *GU_Object2D*
  Restituisce un oggetto geometrico nello spazio 2D che descrive l'insieme di punti ottenuti dalla proiezione nello spazio 2D dell'insieme di punti rappresentato dall'oggetto. Il tipo di oggetto restituito dipende dallo specifico sottotipo considerato.

Operazioni Insiemistiche: Le operazioni insiemistiche (unione, intersezione, differenza) possono essere definite in maniera ovvia su GU_Object, dato che si tratta di un insieme di punti, tuttavia risulta complessa l'interpretazione del risultato in termini di tipi. Ad esempio, l'unione di due curve non è necessariamente una curva, ma può esserlo in casi particolari. Per questo motivo le operazioni insiemistiche ammesse sugli oggetti geometrici vengono definite dopo aver trattato i tipi geometrici.

La funzione PS( ). Dato un oggetto O di un tipo geometrico, O.PS() restituisce l'insieme di punti associati all'oggetto O; questa funzione è introdotta per semplificare le definizioni di questo capitolo.

Commento

Si noti che sul tipo *GU_Object* potrebbero essere definite molte altre proprietà (da quelle metriche a quelle insiemistiche), tuttavia quelle elencate rappresentano le minime necessarie per definire le proprietà dei tipi e per la definizione di vincoli spaziali su oggetti del tipo.

I tipi GU_Object2D e GU_Object3D

Definizione dei valori possibili

I tipi astratti *GU_Object2D* e *GU_Object3D* sono stati introdotti per distinguere esplicitamente i tipi che descrivono oggetti rappresentati da insiemi di punti nello spazio 2D da quelli che nello spazio 3D.

Ridefinizione delle proprietà ereditate

- **coordinateDimension()**
  `self.isKindOf(GU_Object2D)` ⇒ `self.coordinateDimension()=2`
  `self.isKindOf(GU_Object3D)` ⇒ `self.coordinateDimension()=3`

I tipi GU_PrimitiveObject2D e GU_PrimitiveObject3D

Definizione dei valori possibili

Questi tipi astratti rappresentano una generica geometria primitiva nello spazio 2D e 3D rispettivamente e sono introdotti per semplificare la definizione degli aggregati generici.

I tipi GU_Point2D e GU_Point3D (Point)

Definizione dei valori possibili

Un oggetto geometrico dei tipi *GU_Point2D* e *GU_Point3D* è un oggetto zero-dimensionale chiamato "punto" che rappresenta una posizione in uno spazio di coordinate 2D e 3D rispettivamente.

Ridefinizione delle proprietà ereditate

- **boundary()**
  `self.boundary()` = ∅
- **dimension()**
  `self.dimension() = 0`
- **isCycle()**
  `self.isCycle() = true`
- **isSimple()**
  `self.isSimple() = true`
- **planar()**
  `self.isKindOf(GU_Point2D)` ⇒ `self.planar()= self`
  `self.isKindOf(GU_Point3D)` ⇒ `self.planar() = q`,
  dove q ha le seguenti proprietà: `q.isKindOf(GU_Point2D)=true` e q è ottenuto dall'oggetto originale eliminando la coordinata Z dall'oggetto `self`.

I tipi GU_CPCurve2D e GU_CPCurve3D

Definizione dei valori possibili

I tipi *GU_CPCurve2D* e *GU_CPCurve3D* permettono la definizione di un oggetto monodimensionale che corrisponde al concetto intuitivo di curva elementare continua ottenuta “muovendo” con continuità un punto nello spazio, dove quindi non sono ammesse biforcazioni e punti di rottura della continuità. Inoltre non sono ammesse autointersezioni su infiniti insiemi di punti. Esempi di curve elementari corrette sono riportati in Figura 4.2, mentre in Figura 4.3 sono riportati esempi scorretti di curva.

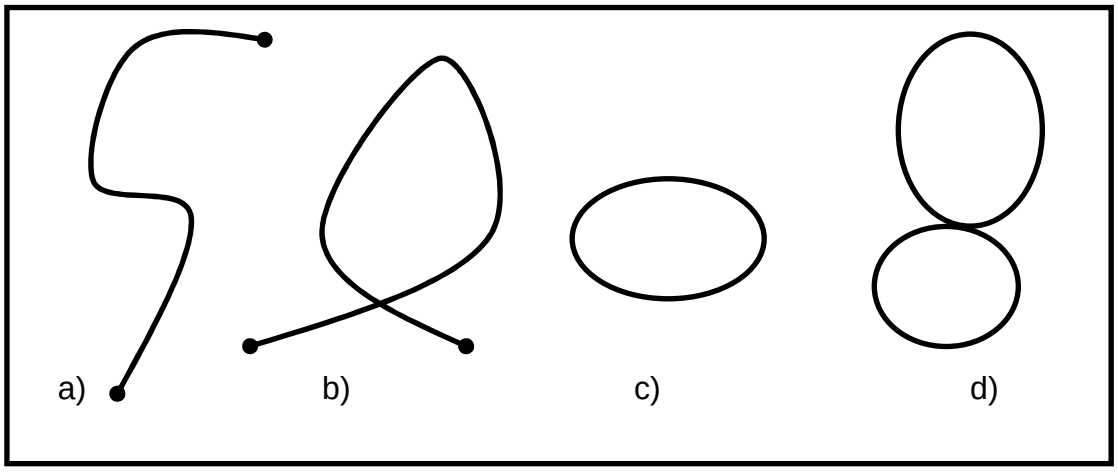


Figura 4.2 - Esempi di curve elementari (*GU_CPCurve2D*).

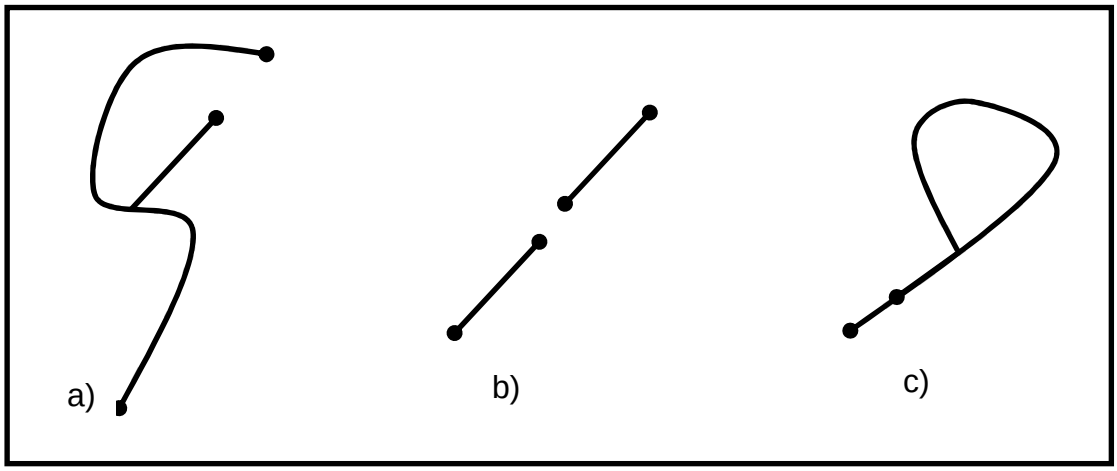


Figura 4.3 – Esempi di oggetti non descrivibili come curve elementari.

In termini matematici dato un intervallo chiuso di numeri reali associati ad un oggetto

$$\texttt{self.Domain} = [a, b] = \{t \in \Re \mid a \le t \le b\}, \text{ con } a<b$$

una curva viene definita come un insieme di punti ottenuto attraverso una funzione *f* continua

$$\texttt{self.f} : [a, b] \rightarrow \Re^n,$$

dove n=2 per curve del tipo *GU_CPCurve2D* e n=3 per curve di tipo *GU_CPCurve3D*.

La curva ammette che esista al più un numero discreto di valori del dominio per i quali la funzione `f` restituisca lo stesso punto dello spazio considerato.

$\forall x1,x2,x3,x4 \in$ self.Domain

((x1<x2 $\wedge$ x2<x3 $\wedge$ x3<x4) $\Rightarrow$ (f([x1,x2]) $\neq$ f([x3,x4]))

dove f([x,y]) indica la porzione di curva ottenuta applicando f all'intervallo [x,y].

Ridefinizione delle proprietà ereditate

- **boundary()**
  La frontiera di una curva viene determinata supponendo che la curva sia definita in uno spazio monodimensionale e quindi coincide con i due punti estremi nelle curve aperte (casi a) e b) di Figura 4.2), mentre la frontiera non esiste se la curva si chiude su se stessa (casi c) e d) di Figura 4.2).
  (self.f(a) = self.f(b)) $\Rightarrow$ self.boundary() = $\varnothing$
  (self.f(a) $\neq$ self.f(b))
  $\Rightarrow$ self.boundary() = {self.f(a), self.f(b)}
- **dimension()**
  self.dimension = 1
- **isCycle()**
  permette di determinare se la curva si richiude su se stessa (casi (c) e (d) di Figura 2.2).
  (self.f(a) = self.f(b)) $\Rightarrow$ self.isCycle() = true
  (self.f(a) $\neq$ self.f(b)) $\Rightarrow$ self.isCycle() = false
- **isSimple()**
  Restituisce true se la curva non passa due volte dallo stesso punto (caso (a) di Figura 4.2) oppure tale punto coincide con i soli estremi della curva (caso (c) di Figura 4.2). Si noti che una curva non semplice può comunque intersecarsi solo in un numero discreto di punti (casi (b) e (d) di Figura 4.2).
  self.isSimple() = true $\Leftrightarrow \forall x_1,x_2 \in$ self.Domain(
  (self.f(x_1)=self.f(x_2)$\wedge x_1 \neq x_2$)$\Rightarrow$(x_1=a$\wedge x_2$=b))
- **planar()**
  La proiezione nello spazio 2D di una curva primitiva 3D in generale genera una curva primitiva dello stesso tipo della curva originale, ma in alcuni casi la curva proiettata genera un oggetto di tipo diverso, ad esempio nei casi seguenti: una curva verticale composta da vertici nei quali cambia solo la coordinata Z genera un punto nel piano, un anello nel piano XZ genera una curva semplice nel piano, una curva semplice con segmenti che si sovrappongono o si intersecano nella proiezione genera un aggregato di curve per descrivere il grafo nel piano.
  self.isKindOf(GU_CPCurve2D) $\Rightarrow$ self.planar() = self
  self.isKindOf(GU_CPCurve3D) $\Rightarrow$ self.planar() = q,
  dove q ha le seguenti proprietà: q.isKindOf(GU_Object2D)=true e q è l'oggetto che descrive l'insieme di punti ottenuto eliminando la coordinata Z da tutti i punti dell'insieme dei punti che descrivono l'oggetto self.

I tipi GU_CPSimpleCurve2D e GU_CPSimpleCurve3D (Composite Simple Curve)

Definizione dei valori possibili

I tipi *GU_CPSimpleCurve2D* e *GU_CPSimpleCurve3D* permettono la definizione di una curva semplice e aperta (caso (a) di Figura 4.2).

Ridefinizione delle proprietà ereditate

- **isSimple()**
  `self.isSimple() = true`

I tipi GU_CPRing2D e GU_CPRing3D (Composite Ring)

Definizione dei valori possibili

I tipi *GU_CPRing2D* e *GU_CPRing3D* permettono la definizione di una curva semplice e chiusa su se stessa, corrispondente al concetto intuitivo di anello (caso (c) di Figura 4.2).

Ridefinizione delle proprietà ereditate

- **boundary()**
  `self.boundary() = ∅`
- **isCycle()**
  `self.isCycle() = true`
- **isSimple()**
  `self.isSimple() = true`

Il tipo GU_CPSurface2D

Definizione dei valori possibili

Un oggetto geometrico definito da questo tipo corrisponde ad una superficie bidimensionale elementare definita nello spazio 2D. Una superficie elementare è definita da un insieme di anelli di tipo *GU_CPRing2D*: un anello `fe` che rappresenta la frontiera esterna della superficie e un insieme di zero o più anelli $Fi = \{fi_1, \ldots, fi_n\}$ che rappresentano le frontiere interne che delimitano gli eventuali buchi della superficie; si noti che poiché un anello non si autointerseca una frontiera non può possedere asole che violerebbero la connessione (definita poi) e la superficie non può degenerare ad una curva aperta (la frontiera esterna composta da un solo segmento percorso in un senso e poi in quello inverso).
La definizione matematica di superficie elementare si basa sulla proprietà di un anello `f` di dividere l'insieme dei punti dello spazio 2D in due regioni (teorema di Jordan): una regione interna chiusa di **area finita** che indichiamo con `Int(f)` e una regione esterna di **area infinita** che indichiamo con `Ext(f)`; entrambe le regioni includono `f`.
Una superficie `S` descritta dall'anello esterno `fe` e dall'insieme degli anelli interni `Fi` risulta costituita dall'insieme di punti dello spazio 2D che soddisfano le seguenti proprietà:

1. La superficie `S` è descritta dai punti interni alla frontiera esterna ed esterni ad ogni frontiera interna, e include le frontiere per garantire la chiusura topologica della superficie:

   $S = Int(fe) \cap Ext(fi_1) \cap \ldots \cap Ext(fi_n)$, con $fi_k \in Fi$, $\forall k \in [1,n]$.

2. Tutti i buchi devono essere contenuti nella regione interna definita dalla frontiera esterna e ogni frontiera interna può toccare la frontiera esterna al più in un solo punto; si noti che un buco che tocca in due punti la frontiera esterna rende la superficie non connessa oppure la coincidenza tra la frontiera esterna e una frontiera interna provoca la degenerazione della superficie ad una curva:

   $\forall fi_k \in Fi\ (Int(fi_k) \subset Int(fe)$
   $\wedge\ ((fi_k.PS() \cap fe.PS() = \emptyset) \vee (|fi_k.PS() \cap fe.PS()|=1))).$

3. Un buco non può essere contenuto in un altro buco o sovrapporsi ad esso. Inoltre due buchi possono toccarsi al più in un punto analogamente al caso precedente:

$\forall fi_k, fi_j \in Fi,\ (fi_k \neq fi_j \Rightarrow (Int(fi_k) \subset Ext(fi_j)$
$\wedge ((fi_k.PS() \cap fi_j.PS()) = \varnothing) \vee (|fi_k.PS() \cap fi_j.PS()| = 1)))).$

4. La superficie S deve avere la sua parte interna connessa, ossia tale che due qualsiasi punti della superficie S (escluse le frontiere) sono connessi da una curva che non attraversa la frontiera. Definiti:

   - C come l'insieme di tutte le curve elementari di tipo *GU_CPCurve2D* definibili nello spazio 2D
   - IS (parte interna di S) = $S - (fe.PS() \cup fi_1.PS() \cup \ldots \cup fi_n.PS())$

     con $fi_k \in Fi,\ \forall k \in [1,n]$

   si ha che

   $\forall p_i, p_j \in IS\ (p_i \neq p_j \Rightarrow$

   $(\exists c \in C\ ((c.PS() \subset IS) \wedge (c.f(a) = p_i) \wedge (c.f(b) = p_j)))$

Nelle Figure 4.4 e 4.5 sono mostrati rispettivamente esempi di superfici corrette e non corrette.

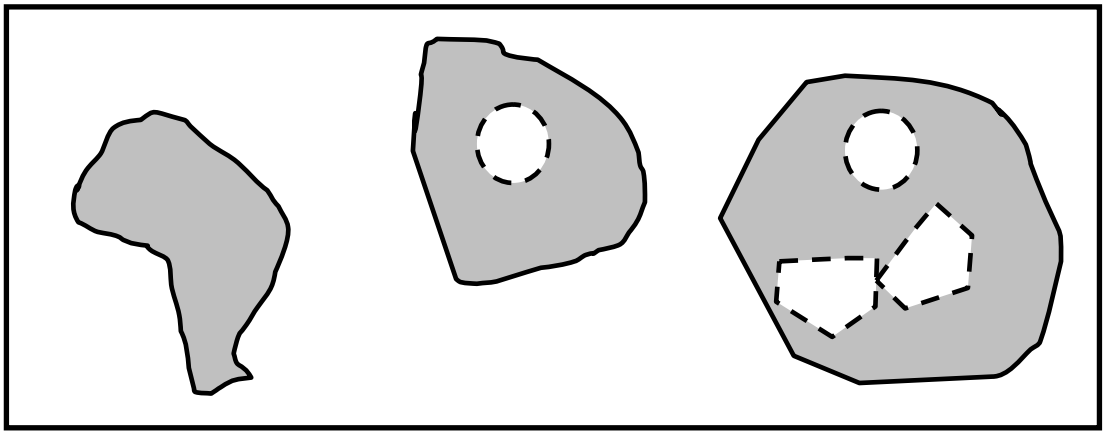


Figura 4.4 – Esempi di superfici (*GU_CPSurface2D*). Le curve tratteggiate rappresentano frontiere interne.

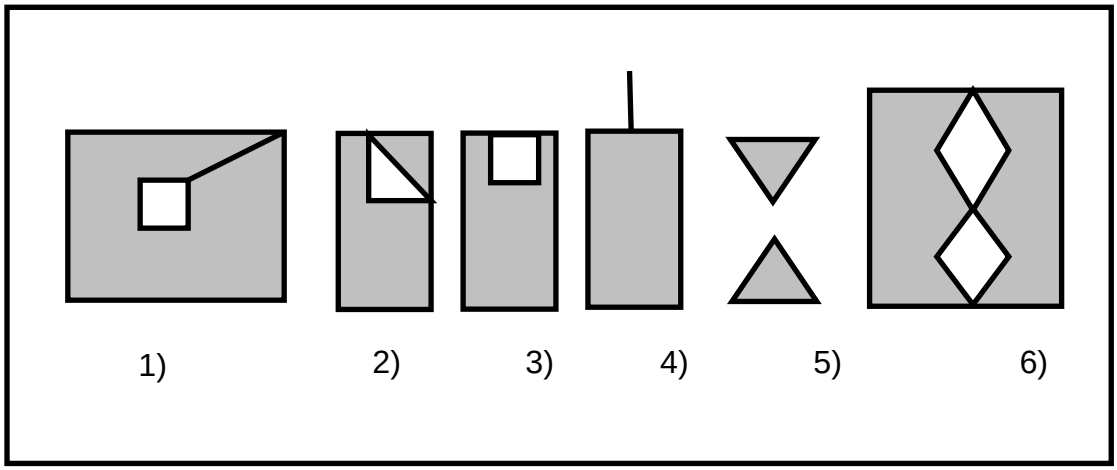


Figura 4.5 – Esempi di geometrie non descrivibili come oggetti del tipo *GU_CPSurface2D.*

Si noti che i poligoni 2, 5 e 6 di Figura 4.5 sono descrivibili come aggregati di due superfici, mentre gli altri richiedono la rimozione di un segmento lineare per essere considerate superfici ammesse.

Ridefinizione delle proprietà ereditate

- **boundary()**
  restituisce un aggregato del tipo *GU_CXRing2D* i cui componenti sono gli anelli che rappresentano le frontiere esterna e interne della superficie.
  $self.boundary().element = \{fe, fi_1, \ldots, fi_k, \ldots, fi_n\}$
  $\quad con\ fi_k \in Fi,\ \forall k \in [1,n]$
- **dimension()**
  $self.dimension( ) = 2$
- **isCycle()**
  una superficie non si può chiudere su se stessa nello spazio 2D.
  $self.isCycle() = false$
- **isSimple()**
  una superficie non può autointersecarsi in uno spazio 2D
  $self.isSimple() = true$

I tipi aggregati generici GU_Aggregate2D e GU_Aggregate3D

Definizione dei valori possibili

I tipi *GU_Aggregate2D* e *GU_Aggregate3D* permettono la definizione di un aggregato, nello spazio 2D e 3D rispettivamente, composto da una collezione di zero o più oggetti geometrici primitivi (anche di tipi diversi) che condividono lo stesso sistema di riferimento che rappresenta quello dell'aggregato. Non sono ammessi aggregati di aggregati. Infine un aggregato generico non impone vincoli alle geometrie dei componenti (possono sovrapporsi e anche coincidere).
Dal punto di vista matematico un aggregato A è interpretato come l'insieme dei punti ottenuto dall'unione degli insiemi dei punti dei singoli oggetti componenti:

$$A.PS() = g_1.PS() \cup \ldots \cup g_n.PS(), \quad \forall g_i \in A.element$$

Nel GeoUML sono stati definiti dei sottotipi dell'aggregato generico allo scopo di restringere i tipi di componenti in base alla dimensione: solo punti nei tipi *GU_CXPoint2D e GU_CXPoint3D, solo curve nei tipi GU_CXCurve2D, GU_CXCurve3D, GU_CXRing2D, GU_CXRing3D, GU_CNCurve2D e GU_CNCurve3D* e infine solo superfici nel tipo *GU_*CXSurface2D. Infine in alcuni tipi

sono imposti dei vincoli sulle relazioni topologiche ammesse tra i componenti.

Ridefinizione delle proprietà ereditate

- **boundary()**
  `self.boundary() = nullo`
- **dimension()**
  l'aggregato generico può contenere oggetti di diversa dimensione e quindi non è possibile associare la dimensione staticamente al tipo come avviene per i suoi sottotipi e pertanto la dimensione dell'aggregato è determinata dall'oggetto di dimensione più grande.
  `self.dimension()= max({g.dimension()|g∈self.element}`
- **Dimensione delle coordinate di un oggetto**
  `self.isKindOf(GU_Aggregate2D)⇒(self.coordinateDimension()=2`
  `∧ ∀g∈self.element (g.coordinateDimension()=2))`
  `self.isKindOf(GU_Aggregate3D)⇒(self.coordinateDimension()=3`
  `∧ ∀g∈self.element (g.coordinateDimension()=3))`
- **isSimple()**
  `self.isSimple() = nullo`
- **isCycle()**
  `self.isCycle = nullo`
- **planar()**
  restituisce un oggetto di tipo *GU_Aggregate2D*
  `self.planar().element = {g.planar()|g∈self.element}`

I tipi GU_CXPoint2D e GU_CXPoint3D (Complex Point)

Definizione dei valori possibili

Un oggetto geometrico dei tipi *GU_CXPoint2D* e *GU_CXPoint3D* è un aggregato di zero o più punti appartenenti tutti al tipo *GU_Point2D* e *GU_Point3D* rispettivamente.

Ridefinizione delle proprietà ereditate

- **boundary()**
  `self.boundary() = ∅`
- **dimension()**
  `self.dimension() = 0`
- **isCycle()**
  `self.isCycle() = true`
- **isSimple()**
  un insieme di punti è semplice se sono tutti geometricamente disgiunti tra loro.
  `self.isSimple() = true`
  $\Leftrightarrow \neg\exists g_i, g_j \in \text{self.element}\ (g_i \neq g_j \wedge (g_i.PS() = g_j.PS()))$

I tipi GU_CXCurve2D e GU_CXCurve3D (Complex Curve)

Definizione dei valori possibili

Un oggetto dei tipi *GU_CXCurve2D* e *GU_CXCurve3D* è un aggregato mono-dimensionale costituito da una collezione di zero o più curve del tipo *GU_CPCurve2D* e *GU_CPCurve3D* rispettivamente che non devono sovrapporsi tra di loro in modo parziale o completo (duplicazione) al fine di mantenere invariante la proprietà di frontiera dell'aggregato.
Questo tipo viene usato per definire curve complesse dove sono ammesse biforcazioni e punti di rottura della continuità, generando curve complesse che possono non essere connesse.

Definita la parte interna di una curva $c \in$ `GU_CPCurve2D` (`GU_CPCurve3D`) come:
`I(c) = c.PS()-c.boundary().PS()`

$\forall c_i, c_j \in \text{self.element}\ (c_i \neq c_j \Rightarrow$
$((I(c_i) \cap I(c_j) = \emptyset) \vee (|(I(c_i) \cap I(c_j))| < \infty)))$

Ridefinizione delle proprietà ereditate

- **boundary()**
  la frontiera di una curva complessa contiene i punti della curva che appartengono alla frontiera di un numero dispari di curve componenti dell'aggregato (regola ***"mod 2 union rule"*** dello standard ISO 19125). Detto P l'insieme di tutti i punti del tipo *GU_Point2D* (*GU_Point3D*) dello spazio 2D (3D):
  `self.boundary() = {p ∈ P | ∃g∈self.element.boundary()`
  `(g.PS()=p.PS() ∧ g.isOddBoundary(self.element))}`
  dove: `g.isOddBoundary(A)` restituisce true se il punto g è frontiera di un numero dispari di curve dell'insieme A.

  La frontiera della curva di Figura 4.6 a) è costituita dai 4 punti estremi anche nel caso in cui l'aggregato è composto da 4 curve semplici convergenti nel punto di intersezione, viceversa la frontiera della curva di Figura 4.5 b) è costituita dai 3

estremi e dal punto di intersezione interna sia nel caso in cui il punto interno sia frontiera di una sola curva o di tre curve. Per analogia al caso a) di Figura 4.5 anche la frontiera della curva di Figura 4.5 c) è costituita dai soli estremi delle curve componenti, mentre nel caso d) la frontiera è vuota poiché tutti i componenti sono cycle.

- **dimension()**
  `self.dimension() = 1`
- **isCycle()**
  `self.isCycle()= true ⇔ ∀g∈self.element (g.isCycle())`
- **isSimple()**
  L'aggregato è semplice se ogni curva componente è semplice e se le curve si toccano tra di loro al più sui punti di frontiera; si noti che questo vincolo impedisce la sovrapposizione della parte interna di due curve componenti, ma anche che una curva tocchi con un proprio punto di frontiera la parte interna di un'altra curva componente.

```
self.isSimple() = true  ⇔
  ∀g∈self.element (g.isSimple()
  ∧(∀gi,gj∈self.element (gi≠gj ⇒
        ((gi.PS() ∩ gj.PS())
          =
         (gi.boundary().PS()∩ gj.boundary().PS()))))))
```

La Figura 4.6 mostra un aggregato semplice (caso c), aggregati non semplici (i casi a e d) con componenti semplici; l'aggregato di Figura 4.6 (caso b) è semplice se è composto da 3 curve che si toccano nel punto interno, mentre non sarà considerato semplice se contiene 2 curve con una che tocca con la propria frontiera la parte interna dell'altra.

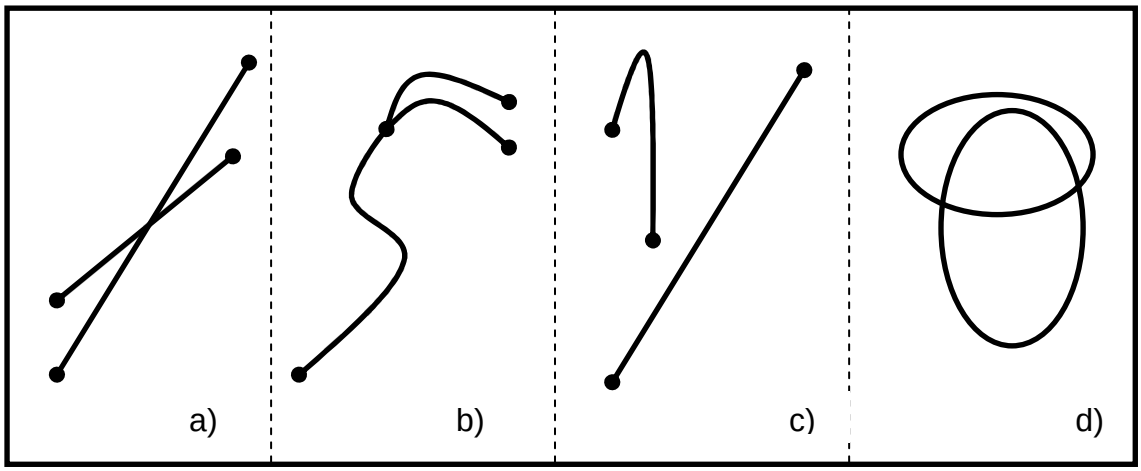


Figura 4.6 – Esempi di aggregati di curve (*GU_CXurve2D*).

Commento

Si noti che NON esiste una corrispondenza biunivoca tra un aggregato di curve inteso come collezione di oggetti geometrici primitivi e l'insieme di punti che lo rappresenta nello spazio poiché lo stesso insieme di punti può corrispondere ad aggregati differenti di oggetti; ad esempio, l'aggregato di figura 4.6.b può essere composto da 2, 3 o più curve primitive. La definizione di frontiera della curva complessa si basa sull'insieme di punti descritto dall'aggregato ed è quindi invariante rispetto alle diverse composizioni di

oggetti dell'aggregato che corrispondono allo stesso insieme di punti dello spazio considerato.

I tipi GU_CXRing2D e GU_CXRing3D (Complex Ring)

Definizione dei valori possibili

I tipi *GU_CXRing2D* e *GU_CXRing3D* sono una specializzazione dei tipi *GU_CXCurve2D e GU_CXCurve3D* rispettivamente. Essi permettono la definizione di un aggregato mono-dimensionale costituito da una collezione di zero o più anelli del tipo *GU_CPRing2D* e *GU_CPRing3D* rispettivamente. Si noti che eredita il divieto di sovrapposizione parziale o totale dalla classe *GU_CXCurve* della corrispondente dimensione, ma non vincola ulteriormente le relazioni topologiche possibili tra i componenti.

Ridefinizione delle proprietà ereditate

- **boundary()**
  `self.boundary()` = ∅
- **isCycle()**
  `self.isCycle()` = true

I tipi GU_CNCurve2D e GU_CNCurve3D (Connected Curve)

Definizione dei valori possibili

I tipi *GU_CNCurve2D* e *GU_CNCurve3D* sono specializzazioni dei tipi *GU_CXCurve2D* e *GU_CXCurve3D* rispettivamente che impongono alla curva complessa la proprietà di connessione della parte interna: due qualsiasi punti della curva complessa sono connessi da una curva elementare contenuta nella curva complessa.
Definito C come l'insieme di tutte le curve elementari (*GU_CPCurve2D /GU_CPCurve3D*)

$$\forall\ p_i, p_j \in \texttt{self.PS()}\ (p_i \neq p_j \Rightarrow (\exists c \in \texttt{C}\ (\texttt{c.PS()} \subset \texttt{self.PS()} \wedge \texttt{c.f(a)}=p_i \wedge \texttt{c.f(b)}=p_j)))$$

Il tipo GU_CXSurface2D (Complex Surface)

Definizione dei valori possibili

Un oggetto del tipo *GU_CXSurface2D* è una superficie complessa costituita da una collezione di zero o più superfici di tipo *GU_CPSurface2D* che sono disgiunte o che al più possono toccarsi solo attraverso punti della frontiera (la superficie complessa è quindi in generale un oggetto non connesso):

Definite la parte interna `I(g)` e la frontiera `F(g)` di una superficie g ∈ `GU_CPSurface2D` come:

`I(g) = g.PS()-g.boundary().PS()` e
`F(g) = g.boundary().PS()`

$$\forall g_i, g_j \in \texttt{self.element}\ (g_i \neq g_j \Rightarrow ((\texttt{I}(g_i) \cap \texttt{I}(g_j) = \emptyset) \wedge (\texttt{F}(g_i) \cap \texttt{F}(g_j) \neq \emptyset) \Rightarrow |\texttt{F}(g_i) \cap \texttt{F}(g_j)| < \infty))$$

Si noti che l'adiacenza su un tratto della frontiera non è ammesso poiché le due superfici sarebbero rappresentabili con un'unica superficie di tipo *GU_CPSurface2D*.

Ridefinizione delle proprietà ereditate

- **boundary()**
  restituisce un aggregato del tipo *GU_CXRing2D* i cui componenti sono gli anelli che rappresentano le frontiere esterna e interne di tutte le superfici componenti dell'aggregato.
  `self.boundary()=self.element.boundary()`
- **dimension**
  `self.dimension() = 2`
- **isCycle**
  La superficie planare non è chiusa.
  `self.isCycle() = false`
- **isSimple**
  Le singole superfici componenti sono per definizione semplici e inoltre la definizione dei vincoli imposti dal tipo garantiscono la proprietà di semplicità dell'aggregato.
  `self.isSimple = true`

La Figura 4.7 mostra superfici complesse composte da due superfici elementari disgiunte (caso a)), adiacenti in un punto (caso b)) e in due punti (caso(c)). La Figura 4.8 mostra due superfici non rappresentabili come superfici complesse, ma come superficie elementare (caso b)) e come aggregato generico (caso a)) nel quale il tratto lineare è una curva distinta dalle due superfici.

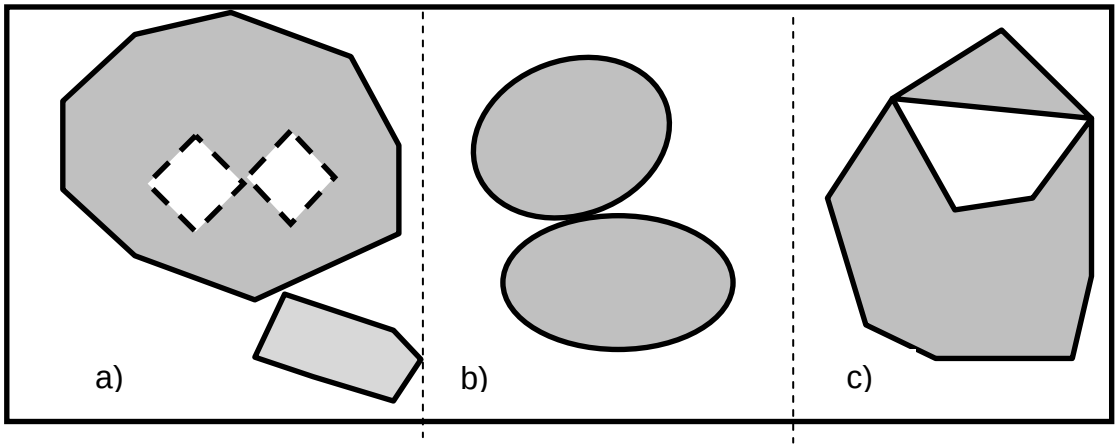


Figura 4.7 – Esempi di superfici complesse (*GU_CXSurface2D*) composte da due superfici elementari

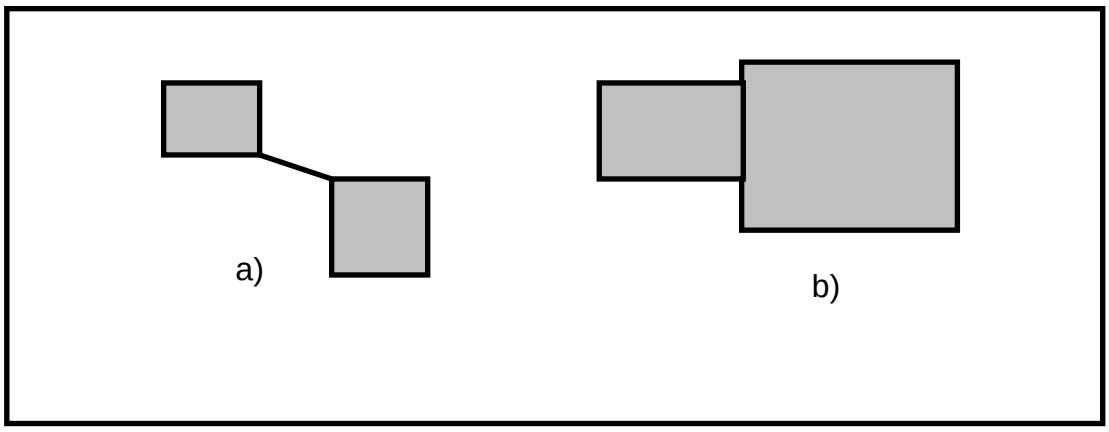


Figura 4.8 – Esempi di geometrie non descrivibili come GU_CXSurface2D

I tipi GU_CPSurfaceB3D/GU_CXSurfaceB3D (Composite/Complex Surface Boundary 3D)

Il GeoUML modella le superfici nello spazio 3D attraverso il concetto di superficie con frontiera in 3D (tipi *GU_CPSurfaceB3D* e *GU_CXSurfaceB3D).* Questi due tipi descrivono la superficie attraverso due attributi geometrici legati tra loro da un vincolo:

- l'attributo "B3D" che in entrambi i tipi descrive la frontiera reale della superficie nello spazio 3D; tale frontiera può essere composta da più anelli ed è definita tramite un aggregato di anelli del tipo *GU_CXRing3D;*
- l'attributo "superficie" che descrive la proiezione planare della superficie nello spazio 2D tramite una superficie primitiva *GU_CPSurface2D* nel tipo *GU_CPSurfaceB3D* e un aggregato di superfici di tipo *GU_CXSurface2D* nel tipo *GU_CXSurfaceB3D.*

Il vincolo che lega i due attributi impone che la frontiera della superficie proiettata coincida con la proiezione planare della frontiera 3D.

Attributi specifici del tipo GU_CPSurfaceB3D

- **superficie**: GU_CPSurface2D
- **B3D:** GU_CXRing3D;

Attributi specifici del tipo GU_CXSurfaceB3D

- **Superficie:** GU_CXSurface2D;
- **B3D:** GU_CXRing3D;

Vincolo sugli attributi

self.B3D.planar().PS() = self.superficie.boundary().PS()

Ridefinizione delle proprietà ereditate

Le proprietà definite su `GU_Object` hanno significato per le geometrie componenti di una superficie di questo tipo, ma non per la geometria composta, pertanto le funzioni boundary(), coordinateDimension(), dimension(), isCycle(), isSimple() e planar() assumeranno il valore nullo**.**

Commento e Esempio

Il vincolo sugli attributi restringe la configurazione di anelli che descrivono la curva B3D a solo quelli che proiettati rimangono anelli che soddisfano i vincoli imposti dall'attributo superficie.

Le superfici B3D trovano molte possibili applicazioni, perché permettono di vedere gli oggetti areali considerandoli nello spazio tridimensionale come semplici anelli (cioè senza la determinazione esatta della superficie tridimensionale delimitata dall'anello stesso), ma permettono allo stesso tempo di definire molte proprietà aggiuntive, che richiedono il riferimento a superfici, quali la copertura di un'area, l'adiacenza, il contenimento di altri oggetti geometrici, ecc.., riferendosi alle superfici 2D delimitate dalle proiezioni di tali anelli.

Un esempio di dichiarazione in GeoUML con riferimento a questo tipo è il seguente:

classe Lago (LAG – 0802)
attributi
componenti spaziali della classe

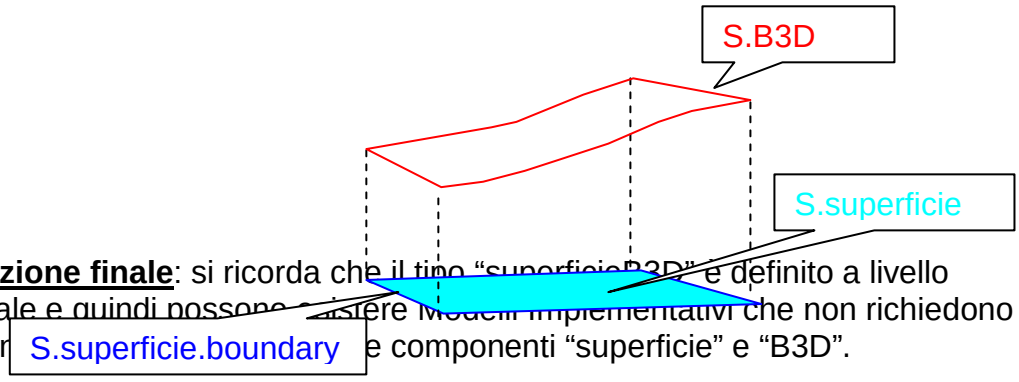
080201 - estensione: GU_CPSurfaceB3D;
...

Si noti che il tipo è una composizione di altri tipi e quindi non sono associate proprietà al tipo nel suo complesso. Nelle relazioni spaziali e in tutte le espressioni di vincoli è necessario far riferimento agli attributi componenti. Questo implica che quando ci si riferisce ad un attributo di tipo *GU_C*SurfaceB3D* va aggiunto `.superficie` oppure `.B3D` a seconda di quale delle due componenti geometriche si voglia considerare; con riferimento all'esempio precedente si dovrà quindi scrivere

"Lago.estensione.superficie" oppure "Lago.estensione.B3D"

per esprimere relazioni o vincoli che fanno riferimento alla componente spaziale della classe Lago.

Nella figura seguente è rappresentato un oggetto di tipo GU_CPSurfaceB3D e sono evidenziate le sue 2 componenti; la figura mostra anche l'uso della funzione boundary() con riferimento alla componente "superficie" dell'oggetto.



Osservazione finale: si ricorda che il tipo "superficieB3D" è definito a livello concettuale e quindi possono esistere modelli implementativi che non richiedono di rappresentare le componenti "superficie" e "B3D".

Le funzioni *gUnion* (unione geometrica) e *gIntersection* degli oggetti geometrici

L'operazione gUnion (per distinguerla dalla union tra oggetti) si applica a due oggetti di uno dei tipi geometrici definiti (o ad uno dei due componenti di una superficie con frontiera in 3D) e produce l'insieme di punti ottenuto dall'unione insiemistica degli insiemi di punti degli oggetti coinvolti. Tale insieme di punti è poi associato ad un oggetto o ad un aggregato di oggetti di uno dei tipi definiti nel modello geometrico del GeoUML. Lo stesso vale per l'operazione gIntersection, vale a dire, si applica a due oggetti di uno dei tipi geometrici definiti (o ad uno dei due componenti di una superficie con frontiera in 3D) e produce l'insieme di punti ottenuto dall'intersezione insiemistica degli insiemi di punti degli oggetti coinvolti. Come per gUnion il risultato viene poi associato ad un oggetto di uno dei tipi geometrici del GeoUML.
In Tabella 4.1 si riportano i tipi che possono essere coinvolti nelle operazioni e nelle tabelle 4.2(a) e 4.2(b) si mostrano i sottotipi del tipo *GU_Object* prodotti dall'operazione di gUnion e gIntersection rispettivamente. Nelle tabelle si riportano solo i codici dei tipi per gli operandi e per il risultato. Si noti che gli oggetti coinvolti in un'esecuzione dell'operazione e il relativo risultato devono appartenere allo stesso spazio (2D o 3D). Alcune caselle delle tabelle indicano la possibilità di generare risultati di tipi diversi e in particolare il tipo aggregato generico quando si effettua l'unione (o l'intersezione) di oggetti di dimensione differente.

Codice tipo	Tipo
∅	Insieme vuoto
P	*GU_Point*D*
C	*GU_CPCurve*D, GU_CPSimpleCurve*D, GU_CPRing*D,*
S	*GU_CPSurface2D*
MP	*GU_CXPoint*D,*
MC	*GU_CXCurve*D, GU_CXRing*D, GU_CNCurve*D*
MS	*GU_CXSurface2D*
A	*GU_Aggregate*D*

Tabella 4.1. Tipi degli operandi delle operazioni di gUnion e gIntersection

gUnion(a,b)

b / a	∅	P	C	S	MP	MC	MS	A
∅	∅	P	C	S	MP	MC	MS	A
P	//////////////	P, MP	C, A	S, A	MP	MC, A	MS, A	A
C	//////////////	//////////////	C, MC	S, A	C, A	C, MC	MS, A	A
S	//////////////	//////////////	//////////////	S, MS	S,A	S, A	S, MS	A
MP	//////////////	//////////////	//////////////	//////////////	MP	MC, A	MS, A	A
MC	//////////////	//////////////	//////////////	//////////////	//////////////	C, MC	MS, A	A
MS	//////////////	//////////////	//////////////	//////////////	//////////////	//////////////	S, MS	A
A	//////////////	//////////////	//////////////	//////////////	//////////////	//////////////	//////////////	A

(a)

gIntersection(a,b)

b / a	∅	P	C	S	MP	MC	MS	A
∅	∅	∅	∅	∅	∅	∅	∅	∅
P	//////	P	P	P	P	P	P	P
C	////////// //////////	////// //////	P, MP, C, MC, A	P, MP, C, MC, A	P, MP	P, MP, C, MC, A	P, MP, C, MC, A	P, MP, C, MC, A
S	//////////	//////	////////////////	qualsiasi	P, MP	P, MP, C, MC, A	qualsiasi	qualsiasi
MP	////////// //////////	////// //////	//////////////// ////////////////	//////////////////// ////////////////////	P, MP	P, MP	P, MP	P, MP
MC	////////// //////////	////// //////	//////////////// ////////////////	//////////////////// ////////////////////	////////////// //////////////	P, MP, C, MC, A	P, MP, C, MC, A	P, MP, C, MC, A
MS	//////////	//////	////////////////	////////////////////	//////////////	////////////////	qualsiasi	qualsiasi
A	//////////	//////	////////////////	////////////////////	//////////////	////////////////	////////////////	qualsiasi

(b)

Tabella 4.2 Tipi dell'oggetto prodotto dall'operazione gUnion (a) e gIntersection (b).

Le relazioni topologiche sugli oggetti geometrici.

Per descrivere le relazioni spaziali tra gli oggetti, in particolare quando si vogliono specificare i vincoli di integrità di natura geometrica in uno schema GeoUML, è necessario utilizzare un insieme di relazioni topologiche di riferimento.
Le relazioni topologiche del GeoUML sono definite utilizzando i concetti di parte interna, frontiera e parte esterna di un oggetto geometrico; dato un oggetto geometrico a di tipo *GU_Object* si definiscono:

1. parte interna *I(a)*: è l'insieme di punti `a.PS()- a.boundary.PS()`(è l'insieme di punti dell'oggetto che non appartengono alla sua frontiera)
2. parte esterna *E(a)*: è l'insieme dei punti dello spazio che non appartengono all'oggetto stesso.

L'insieme fondamentale di relazioni topologiche utilizzato in GeoUML, chiamato REL_{topo}, è composto dalle relazioni Disjoint (DJ), Touch (TC), In (IN), Contains (CT), Equals (EQ) e Overlaps (OV). Questo insieme possiede le seguenti caratteristiche:

- le relazioni che lo compongono sono mutuamente esclusive, cioè se tra due oggetti geometrici vale la relazione R, allora non vale nessuna altra relazione dell'insieme;
- l'insieme è completo, cioè, dati due oggetti geometrici in una certa situazione spaziale esiste sempre una relazione dell'insieme che è vera in quella situazione;
- le relazioni si applicano ad oggetti dello stesso tipo o di tipi differenti;

Le relazioni dell'insieme REL_{topo} non specificano la dimensione del risultato e sono applicabili a tutti gli oggetti geometrici del GeoUML ad eccezione dell'aggregato generico (perchè su questo non è definito il concetto di frontiera); inoltre, nel caso delle superfici B3D devono essere applicate specificando uno degli attributi componenti il tipo. Se le relazioni topologiche sono applicate a tipi aggregati, esse non coinvolgono singolarmente i componenti dell'aggregato, ma si applicano all'insieme di punti dell'aggregato interpretato come il risultato dell'unione dei punti dei componenti (ad esempio, la relazione "Overlaps" è soddisfatta da un aggregato se almeno un componente dell'aggregato la soddisfa).
Inoltre le relazioni dell'insieme REL_{topo} sono valutabili solamente tra oggetti definiti nello stesso spazio (2D o 3D); non è ammesso viceversa il confronto tra oggetti definiti in spazi differenti.

Definizione dell'insieme di relazioni REL_{topo}

Detti `a` e `b` due oggetti geometrici di un qualsiasi tipo ad eccezione dei tipi `GU_Aggregate2D, GU_Aggregate3D, GU_CPSurfaceB3D` e `GU_CXSurfaceB3D`:

DJ: $a.Disjoint(b) \equiv_{def} (a.PS() \cap b.PS() = \varnothing)$

TC: $a.Touch(b) \equiv_{def} (I(a) \cap I(b) = \varnothing) \wedge (a.PS() \cap b.PS() \neq \varnothing)$

IN: $a.In(b) \equiv_{def} (a.PS() \cap b.PS() = a.PS()) \wedge (a.PS() \cap b.PS() \neq b.PS()) \wedge (I(a) \cap I(b) \neq \varnothing)$

CT: $a.Contains(b) \equiv_{def} b.in(a)$

EQ: $a.Equals(b) \equiv_{def} (a.PS() \cap b.PS() = a.PS()) \wedge (a.PS() \cap b.PS() = b.PS())$

OV: $a.Overlaps(b) \equiv_{def} (I(a) \cap I(b) \neq \varnothing) \wedge (a.PS() \cap b.PS() \neq a.PS())$

$\wedge$ (a.PS() $\cap$ b.PS() $\neq$ b.PS())

L'insieme minimo e completo REL_{topo} è arricchito con le relazioni `Intersects` e `Cross` (tra oggetti lineari) derivabili dalle altre ma di uso comune:

INT: a.Intersects(b) $\equiv_{def}$ $\neg$ a.Disjoint(b)

CR: a.Cross(b)$\equiv_{def}$ a.Overlaps(b) $\wedge$ (a.PS() $\cap$ b.PS()).dimension()=0

Commento

Con riferimento alle relazioni definite si può osservare che:

- la relazione DJ impedisce punti in comune tra oggetti, mentre tutte le altre prevedono che i due oggetti abbiano punti in comune;
- se i punti in comune non sono punti interni degli oggetti, allora la relazione è Touch, o adiacenza. La definizione considera non solo i casi ovvi di adiacenza, in cui solo punti della frontiera sono in comune, ma anche i casi più complessi, nei quali i punti di frontiera di un oggetto sono anche punti interni dell'altro. Questo comporta che, ad esempio, una curva contenuta nella frontiera di una superficie sia un caso possibile per la relazione TC. La relazione TC sarà sempre falsa quando entrambi gli oggetti sono del tipo GU_Point*D;
- il contenimento IN (CT) corrisponde intuitivamente al concetto di contenimento insiemistico con l'eccezione del caso in cui un oggetto è contenuto della frontiera dell'altro come citato al precedente punto (relazione Touch) o è uguale all'altro (relazione Equals);
- nel caso di OV i due oggetti hanno punti interni in comune (quindi non sono in Touch), ma non sono in relazione di contenimento o uguaglianza (quindi ambedue gli oggetti hanno una parte che è fuori dalla parte che hanno in comune). La relazione OV sarà sempre falsa quando uno o entrambi gli oggetti sono punti;
- la relazione CR è una specializzazione della relazione OV che si applica solo ad oggetti dei tipi GU_C*Curve*D e verifica che la dimensione dell'intersezione sia puntiforme;
- le relazioni OV, DJ, CR, EQ, TC sono simmetriche (ad esempio, a.Touch(b) è uguale a b.Touch(a);
- la relazione DJ tra due oggetti è sempre vera se la geometria di almeno uno dei due oggetti è vuota, mentre le altre relazioni topologiche sono sempre false in presenza di almeno una geometria vuota.

Le relazioni dell'insieme REL_{topo} (eccetto Equals e Contains) sono illustrate in Figura 4.9, dove in ogni colonna mostra la stessa relazione topologica applicata a oggetti di tipo diverso e in ogni riga mostra le diverse relazioni si applichino alla stessa coppia di tipi di oggetti.

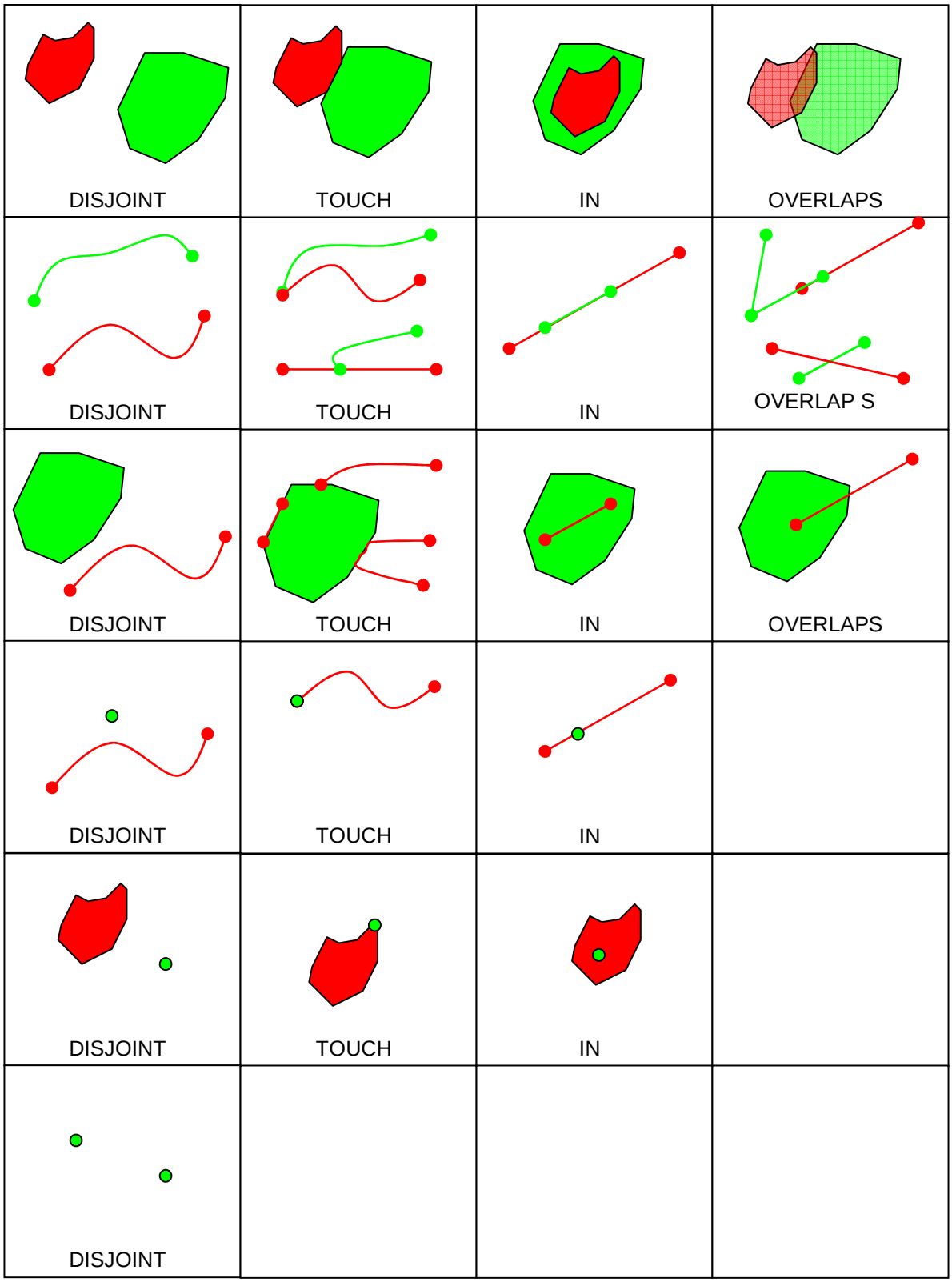


Figura 4.9 – Esempi di relazioni topologiche su diversi tipi di oggetti geometrici.

Attributi dipendenti dalle geometrie

Introduzione

Gli attributi dipendenti dalla geometria sono attributi il cui valore è una funzione dei punti appartenenti ad un attributo geometrico di un oggetto applicativo. Esistono tre varianti dell'attributo dipendente dalla geometria: l'attributo a tratti, a sottoaree e ad eventi. Di seguito si definiscono l'attributo a tratti e a sottoaree dipendenti rispettivamente da una geometria lineare e areale e l'attributo a eventi dipendente da entrambe le geometrie.

Commento e Esempio

Ad esempio, se consideriamo l'attributo "sede" di una strada con un attributo lineare "percorso", il valore di tale attributo non dipende solamente dalla strada considerata ma anche dal punto sul percorso preso in considerazione. I punti del percorso possono essere raggruppati in zone dotate di valore omogeneo della "sede" dette ***tratti***.

Se consideriamo lo stesso esempio immaginando una rappresentazione areale della strada, i punti dotati di valore omogeneo della "sede" costituiscono delle ***sottoaree***.

Le definizioni seguenti si applicano a diversi tipi geometrici e quindi per fattorizzare la formulazione si usano le seguenti convenzioni:

- il carattere asterisco nel nome di un tipo geometrico indica tutti i valori possibili che potrebbero stare in quella posizione (ad esempio, GU_Object*D significa un qualsiasi oggetto in due e tre dimensioni)
- l'indicazione di un tipo geometrico in una definizione significa che la stessa definizione si applica a quel tipo e a qualsiasi sua specializzazione.

Attributo a tratti

Data una classe X contenente un attributo geometrico g di tipo lineare, è possibile definire su g un attributo a tratti A di dominio D_A che descrive una proprietà che dipende dalla geometria di g. Tale definizione si basa sulla parola chiave *aTratti* inserita nella sezione *attributi di questa componente spaziale* della componente spaziale di riferimento, come evidenziato nel seguente esempio.

```
classe X (abbreviazioni)
     ....
     componenti spaziali della classe
          g: GU_C*Curve*D;
               attributi di questa componente spaziale
                    A [min..max]: DA aTratti su g;
          …
```

L'attributo a tratti ha un nome univoco nell'ambito degli attributi della componente spaziale sulla quale è definito, oltre al codice e al codice alfanumerico opzionale. Il dominio D_A dell'attributo a tratti può essere un dominio di base, un dominio enumerato o un dominio enumerato gerarchico senza attributi aggiuntivi di dominio di base (è escluso il dominio DataType).

La definizione del significato di queste tipologie di attributi si basa sulla sostituzione

della forma dichiarativa

```
A [min..max]: DA aTratti su g;
```

con le due funzioni (o metodi) seguenti:

```
ValoreDi_A(p: GU_Point*D): Set(DA)
TrattiDi_A(cond: String): Set (GU_CXCurve*D)
```

Queste due funzioni possono essere immaginate come dichiarate implicitamente e possono essere impiegate per specificare sulle classi alcune proprietà aggiuntive come accade nella specifica dei vincoli di integrità; la loro sostituzione permette di operare come se lo schema dell'esempio precedente fosse scritto nel modo seguente:

Rappresentazione in GeoUML di base + metodi

```
classe X (codiceAlfanumericoX – codX)
     attributi
     componenti spaziali della classe
          g: GU_C*Curve*D;
     metodi
     ValoreDi_A(p: GU_Point*D): Set(DA)
     TrattiDi_A(cond: String): Set (GU_CXCurve*D)
```

La funzione `ValoreDi_A` valuta il valore dell'attributo a tratti in un punto dell'attributo geometrico g. Riceve come parametro un punto P definito nello stesso spazio (2D/3D) dell'attributo g e se il punto appartiene all'attributo geometrico g la funzione restituisce un valore del dominio D_A se la cardinalità è 1..1, oppure un insieme di valori se la cardinalità massima è * (non è possibile assegnare valori diversi da 1 e * alla cardinalità massima). Infine nel caso di assenza di valore (attributo opzionale ossia con cardinalità minima = 0) restituisce il valore *nullo;* si noti che il valore *nullo* è restituito anche nel caso in cui il punto non appartenga al valore dell'attributo geometrico g oppure quando l'attributo g è nullo.
Un tratto è un oggetto geometrico del tipo GU_CXCurve*D (un tratto non può contenere punti isolati) della stessa dimensione dell'attributo geometrico g ed è associato ad un valore $v \in D_A$, pertanto un tratto è costituito dall'unione di tutti i punti P della geometria g per i quali `ValoreDi_A(P)=v.`
Da queste definizioni si deduce che:

- un tratto corrisponde in generale ad una curva complessa e non connessa (il tratto può contenere biforcazioni a causa di autointersezioni o intersezioni tra componenti della geometria);
- nel caso di attributo opzionale può esistere un tratto che raccoglie tutti i punti nei quali l'attributo ha valore *nullo*;
- due tratti diversi possono sovrapporsi ad esempio, quando una porzione della geometria g condivide due o più valori dell'attributo (solo se la cardinalità massima è *) o intersecarsi (in presenza di autointersezioni o intersezioni tra componenti della geometria);
- l'unione (gUnion) di tutti gli insiemi di punti dei tratti definiti su g corrisponde alla geometria di g.

La funzione `TrattiDi_A` restituisce il valore nullo se il valore dell'attributo geometrico g è nullo.
La funzione `TrattiDi_A(condizione di selezione),` data una particolare condizione di selezione restituisce l'insieme dei tratti che soddisfano tale condizione; la clausola di selezione è una formula proposizionale del tipo [not](α_1 opLogico … α_i … opLogico α_n) con opLogico$\in${AND,OR} e α_i è una formula atomica del tipo (A op cost),

(A = nullo) o (A = non nullo) dove A è l'attributo a tratti, op ∈ {=, <>, >, ≥, <, ≤} e cost è un valore diverso dal valore nullo. Una condizione vuota corrisponde alla costante true e pertanto saranno restituiti tutti i tratti definiti sulla geometria g, mentre una condizione sempre falsa restituirà l'insieme vuoto.
E' possibile definire l'attributo a tratti anche sulla frontiera di una superficie di tipo GU_C*Surface2D e GU_C*SurfaceB3D; nel primo caso si definirà un attributo aTratti sul contorno 2D dell'attributo geometrico, mentre per la superficie con frontiera in 3D esistono due possibilità:

- l'attributo è definito sulla curva (componente B3D del tipo) che rappresenta la frontiera della superficie nello spazio 3D ed è descritto come un attributo aTratti sul contorno 3D della geometria,
- l'attributo è definito sulla frontiera della superficie (componente superficie del tipo) ottenuta dalla proiezione nello spazio 2D della componente B3D del tipo ed è descritto come un attributo aTratti sul contorno 2D della geometria. In questi casi sintatticamente il vincolo si esprime come segue, mentre il suo significato è invariato; si noti che le funzioni ValoreDi_A() e TrattiDi_A() sono definite nel template sullo specifico contorno 2D o 3D dell'attributo considerato.

Variante sul contorno

```
classe X
  ...
   componenti spaziali della classe
    g: GU_C*Surface2D oppure GU_C*SurfaceB3D;
    attributi di questa componente spaziale
     A[min..max]: DA aTratti sul contorno 2D|3D di g;
```

Commento e Esempio

Facendo riferimento all'esempio precedente, la seguente definizione associa alle strade un attributo di sede che è a tratti sul percorso.

classe Strada (STR – 0501)
attributi
componenti spaziali della classe
050101 - percorso: GU_CPCurve3D;
attributi di questa componente spaziale
050102 - sede: *Enum* TIPO_SEDE *aTratti su*
percorso;
dominio TIPO_SEDE (TSED – 9977)
01 propria
....

Nota bene: in un Modello Implementativo un attributo a tratti può essere realizzato sia materializzando le geometrie che costituiscono il singolo tratto e associandogli il valore di attributo, sia utilizzando un'ascissa curvilinea per indicare le porzioni della componente spaziale sulle quali l'attributo ha un certo valore. Il Data Product conterrà nella sua struttura fisica le informazioni adeguate al tipo di realizzazione prescelta.
Le funzioni utilizzate qui per definire la semantica dei tratti, che verranno impiegate anche nei prossimi capitoli, ad esempio per esprimere i vincoli, possono essere implementate su entrambe queste materializzazioni in forma generale e quindi un Data Product non ha bisogno di contenere l'implementazione delle funzioni.
Queste considerazioni valgono anche per gli attributi a eventi e a sottoaree; per quest'ultimi però esiste solo la realizzazione tramite materializzazione delle geometrie, e non quella tramite ascissa curvilinea.

Attributo a eventi

La definizione di un attributo a eventi segue regole simili a quello a tratti.
Data una classe X contenente un attributo geometrico g di tipo lineare o di tipo areale (escluse le superfici GU_C*SurfaceB3D), è possibile definire su g un attributo a eventi A di dominio D_A che descrive una proprietà che dipende dalla geometria di g. Tale definizione si basa sulla parola chiave *aEventi su* inserita nella sezione *attributi di questa componente spaziale* della componente spaziale di riferimento, come evidenziato nel seguente esempio

```
classe X
     ...
     componenti spaziali della classe
          g: GU_C*Curve*D oppure GU_C*Surface2D;
               attributi di questa componente spaziale
               A[0..max]: DA aEventi su g;
          …
```

L'attributo a eventi ha un nome univoco nell'ambito degli attributi della componente spaziale sulla quale è definito, oltre al codice e al codice alfanumerico opzionale. Il dominio D_A può essere un dominio di base, un dominio enumerato o un dominio enumerato gerarchico senza attributi aggiuntivi di dominio di base (è escluso qualsiasi dominio DataType).
Un evento è un punto della geometria a cui è associato un valore $v \in D_A$, non tutti i punti della geometria hanno associato un evento (cardinalità minima sempre 0) e possono esistere punti a cui sono associati più eventi (cardinalità massima *).

La definizione del significato di queste tipologie di attributi si basa, in maniera analoga a quanto visto per i tratti, sulla sostituzione della forma dichiarativa precedente con una funzione, come nell'esempio seguente

```
Rappresentazione in GeoUML di base + metodi
     classe X (codiceAlfanumericoX – codX)
          attributi
          componenti spaziali della classe
               codg - g: GU_C*Curve*D oppure GU_C*Surface2D;
          metodi
          EventiDi_A(cond: String): Set(GU_CXPoint*D)
```

La funzione `EventiDi_A` data una particolare condizione di selezione restituisce l'insieme dei punti che soddisfano tale condizione; la clausola di selezione è una formula proposizionale del tipo [not](α_1 opLogico …α_i … opLogico α_n) con opLogico$\in${AND,OR} e α_i è una formula atomica del tipo (A op cost), dove A è l'attributo a eventi, op $\in$ {=, <>, >, ≥, <, ≤} e cost è un valore diverso da nullo. Una condizione vuota corrisponde alla costante true e pertanto saranno restituiti tutti i punti della geometria g sui quali è stato definito un evento.
La funzione `EventiDi_A` restituisce il valore nullo se il valore dell'attributo geometrico g è nullo.

Attributo a sottoaree

Analogamente all'attributo a tratti, l'attributo a sottoaree associa un valore di un dominio a diverse sottoaree di un attributo geometrico areale.

Commento e Esempio

Ad esempio si pensi a una classe "strada" dotata di un attributo geometrico areale "estensione" e all'attributo "sede" che descrive il tipo di sede di tale estensione. In questo esempio potremmo definire un attributo a sottoaree "sede" che associa il valore della sede ai diversi punti dell'estensione della strada; una "sottoarea" è una porzione dall'estensione alla quale è associato un particolare valore dell'attributo.

Data una classe X contenente un attributo geometrico g di tipo areale, è possibile definire su g un attributo a sottoaree A di dominio D_A che descrive una proprietà che dipende dalla geometria di g. Tale definizione si basa sulla parola chiave *aSottoaree su* inserita nella sezione *attributi di questa componente spaziale* della componente spaziale di riferimento, come evidenziato nel seguente esempio

```
classe X (codiceAlfanumericoX – codX)
     ...
     componenti spaziali della classe
        g: GU_C*Surface2D oppure GU_C*SurfaceB3D;
          attributi di questa componente spaziale
          A[min..max]: DA aSottoaree su g;
          …
```

L'attributo a sottoaree ha un nome univoco nell'ambito degli attributi della componente spaziale sulla quale è definito, oltre al codice e al codice alfanumerico opzionale. Il dominio D_A dell'attributo a tratti può essere un dominio di base, un dominio enumerato o un dominio enumerato gerarchico senza attributi aggiuntivi di dominio base (è escluso qualsiasi dominio DataType).

Il significato dell'attributo a sottoaree è definito seguendo le stesse regole di sostituzione con due metodi applicata nel caso dell'attributo a tratti, come nel seguente esempio

Rappresentazione in GeoUML di base + metodi

classe X (CodiceAlfanumericoX – codX)
attributi
componenti spaziali della classe
codg - g: GU_C*Surface2D oppure GU_C*SurfaceB3D;
metodi
ValoreDi_A(p: GU_Point2D): Set(D_A)
SottoareeDi_A(cond: String): Set(GU_CXSurface2D)

La funzione `ValoreDi_A` valuta il valore dell'attributo a sottoaree in un punto dell'attributo geometrico `g`. Riceve come parametro un punto `P` dello spazio 2D e se il punto appartiene all'attributo geometrico g la funzione restituisce un valore del dominio D_A se la cardinalità è 1..1, oppure un insieme di valori se la cardinalità massima è * (non è possibile assegnare valori diversi da 1 e * alla cardinalità massima) e infine nel caso di assenza di valore (attributo opzionale ossia con cardinalità minima =0) restituisce il valore *nullo*; si noti che il valore *nullo* è restituito anche nel caso in cui il punto non appartenga alla geometria oppure quando la geometria è nulla.
Una sottoarea è un oggetto geometrico del tipo classe GU_CXSurface2D (una sottoarea non può mai contenere punti isolati o curve) ed è associata ad un valore $v \in D_A$, pertanto una sottoarea è costituita dall'unione di tutti i punti `P` della geometria g per i quali `ValoreDi_A(P)=v.`

Da queste definizioni si deduce che:
- una sottoarea corrisponde in generale ad una superficie complessa non connessa;
- nel caso di attributo opzionale può esistere una sottoarea che raccoglie tutti i punti nei quali l'attributo è *nullo*;
- due sottoaree diverse possono sovrapporsi ad esempio, quando una porzione della geometria g condivide due o più valori dell'attributo (solo se la cardinalità massima è *)
- l'unione di tutte le sottoaree definite su g corrisponde alla geometria di g.

La funzione `SottoareeDi_A` restituisce il valore nullo se il valore dell'attributo geometrico g è nullo.
La funzione `SottoareeDi_A(condizione di selezione)`data una particolare condizione di selezione restituisce l'insieme delle sottoaree che soddisfano tale condizione; la clausola di selezione è una formula proposizionale del tipo [not](α_1 opLogico …α_i … opLogico α_n) con opLogico$\in${AND,OR} e α_i è una formula atomica del tipo (A op cost) (A = nullo) o (A = non nullo) dove A è l'attributo a sottoaree, op $\in$ {=, <>, >, ≥, <, ≤} e cost è un valore diverso da nullo. Una condizione vuota corrisponde alla costante true e pertanto saranno restituite tutte le sottoaree definite sulla geometria g, mentre una condizione sempre falsa restituirà l'insieme vuoto.

Commento e Esempio

Il seguente esempio in GeoUML è perfettamente analogo a quello visto precedentemente, ma lo svolge interpretando la sede come sottoaree di una rappresentazione areale della strada.

classe Strada (STR – 0501)
attributi
componenti spaziali della classe
050101 - estensione: GU_CPSurface2D;
attributi di questa componente spaziale

05010101 - SedeDiStrada: *Enum* TIPO_SEDE *aSottoaree su* estensione

dominio TIPO_SEDE (TSED – 9977) …

Sottoaree e B3D

Nel caso in cui la componente spaziale di un oggetto sia di tipo SurfaceB3D le sottoaree sono definite sulla componente “superficie”, e quindi sono delle superfici 2D come rappresentato in figura 5.1

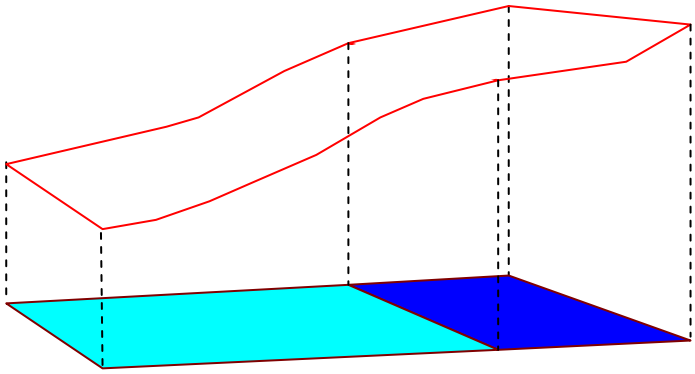


Figura 5.1 – Un oggetto di tipo B3D con 2 sottoaree

Si ricorda che la rappresentazione indicata è a livello concettuale e serve a determinare univocamente l’interpretazione degli oggetti ma permette di creare diversi modelli implementativi.

Vincoli di integrità spaziale

Introduzione

I vincoli di integrità spaziale esprimono condizioni che devono essere soddisfatte dalle componenti spaziali delle istanze delle classi alle quali fanno riferimento.

I vincoli costituiscono un aspetto importante della conformità di un Data Product a una specifica e quindi il loro significato non deve essere ambiguo. Per questo motivo viene fornita una definizione formale del significato dei vincoli basata su un insieme di regole di traduzione nel linguaggio OCL (object constraint language): ***in situazioni in cui il significato intuitivo di un vincolo appare ambiguo è necessario ricorrere a tale definizione per stabilire l'interpretazione corretta.***

Dato che le regole di traduzione in OCL sono molte e faticose da leggere, in questo capitolo vengono illustrati i vincoli in linguaggio naturale, in modo sufficiente a interpretarne il significato generale e la potenzialità espressiva e a capire i capitoli successivi.

Le regole di traduzione in OCL vengono illustrate in questo capitolo a scopo esemplificativo solo per il primo tipo di vincolo, mentre per tutti gli altri si rimanda all'appendice A.

Esistono due famiglie di vincoli di integrità spaziale: i vincoli topologici e i vincoli di composizione.

Vincoli topologici

I vincoli topologici utilizzano le relazioni topologiche dell'insieme REL_{topo}, definite su tutti gli oggetti geometrici istanziabili dei tipi geometrici GeoUML, purché abbiano definito il metodo boundary, quindi gli attributi geometrici di tipo aggregato generico non possono essere coinvolti nei vincoli in quanto ad essi non si applicano le relazioni topologiche.

Esistono 3 categorie di vincoli topologici:

1. i vincoli esistenziali
2. i vincoli su unione
3. i vincoli universali

Per ognuna di queste categorie esistono una versione di base e diverse varianti che permettono di descrivere condizioni più articolate, aggiungendo alla versione base una combinazione dei seguenti elementi:

- selezione sugli oggetti delle classi coinvolte nel vincolo
- applicazione delle funzioni boundary() e planar() alle componenti spaziali degli oggetti coinvolti nel vincolo
- uso di una associazione tra le classi vincolata e vincolante nella formulazione del vincolo
- uso di attributi a tratti, a eventi o a sottoaree nel vincolo

Nel seguito la presentazione dei vincoli topologici procede secondo lo schema seguente:

- prima di tutto viene illustrato dettagliatamente il vincolo topologico esistenziale nella versione base,
- poi vengono illustrate alcune regole generali che valgono nell'espressione dei vincoli,
- poi viene definito, con riferimento al vincolo esistenziale in versione base, il metodo di formalizzazione tramite traduzione in OCL che viene applicato estensivamente in Appendice A,
- poi vengono definite le varianti del vincolo esistenziale,
- poi viene definito il vincolo su unione, senza rispiegare le varianti, che sono le stesse del vincolo esistenziale,
- infine viene definito il vincolo universale, senza rispiegare le varianti, che sono le stesse del vincolo esistenziale.

Vincolo topologico esistenziale di base

Il vincolo topologico esistenziale di base richiede, per ogni oggetto *x* della classe vincolata *X*, l'esistenza di almeno un oggetto *y* della classe vincolante *Y*, tale che un attributo geometrico *f* di *y* si trovi in una particolare relazione topologica con un attributo geometrico *g* di *x*.

Esempio

Descrizione del vincolo in linguaggio naturale:

per ogni ElementoStradale (classe vincolata) deve esistere un'AreaStradale (classe vincolante) la cui Estensione (componente spaziale della classe vincolante) contiene il Percorso (componente spaziale della classe vincolata) dell'ElementoStradale stesso.

Definizione testuale del vincolo:

vincolo ElementoStradale.Percorso (IN) *esiste* AreaStradale.Estensione

tale vincolo opera su classi e componenti spaziali che devono essere definite nel modo seguente:

classe AreaStradale (…)
attributi
componenti spaziali della classe
… Estensione: GU_CPSurface2D
classe ElementoStradale (…)
attributi
componenti spaziali della classe
… Percorso: GU_CPCurve2D)

Gli aspetti che caratterizzano il vincolo sono quindi:

- la ***classe vincolata*** e il suo attributo geometrico (AreaStradale e Estensione nell'esempio),
- la ***classe vincolante*** e il suo attributo geometrico (ElementoStradale e Percorso nell'esempio),
- la ***relazione topologica*** (IN nell'esempio).

Rappresentazione grafica del vincolo:
La rappresentazione grafica dello stesso esempio è mostrata nel Diagramma 6.1, che sostanzialmente contiene gli stessi elementi della definizione testuale, ma obbliga a dare un nome al singolo vincolo (in figura è chiamato *VT_ElementoStradale*).

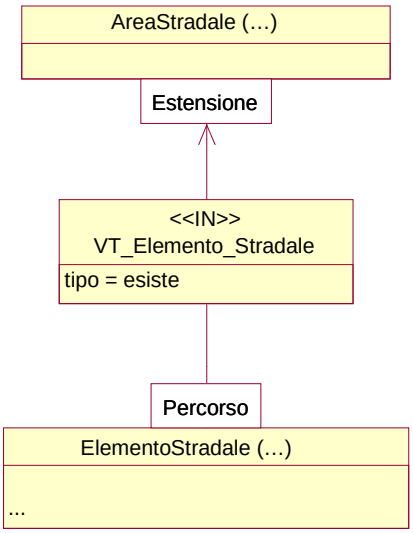


Diagramma 6.1 – Esempio di vincolo topologico esistenziale

Regole generali per la formulazione dei vincoli

Per tutti i vincoli formulabili in GeoUML valgono le seguenti regole generali:

- **Unicità dello spazio per l'applicazione delle relazioni**: i tipi di tutte le geometrie coinvolte in un vincolo devono appartenere allo stesso spazio (2D o 3D) in quanto le relazioni topologiche non sono definite tra oggetti appartenenti a spazi differenti (in alcuni casi per soddifare questa regola vedremo che si può usare la funzione planar() che cambia lo spazio di riferimento di una geometria);
- **Disgiunzione di relazioni topologiche**: la relazione topologica del vincolo può essere sostituita in generale da una disgiunzione di relazioni topologiche elementari, cioè da diverse relazioni poste in OR logico tra loro (ad esempio "DJ oppure TC", indicato come DJ|TC);
- **Gerarchie di ereditarietà:** in presenza di gerarchie di ereditarietà la definizione di un vincolo tra due classi implica che sia implicitamente applicato anche a tutte le sottoclassi (dirette e indirette) della classe vincolata e che per ogni loro oggetto siano coinvolti per la verifica gli oggetti della classe vincolante e quelli di tutte le sue sottoclassi (dirette e indirette). Inoltre la definizione del vincolo può far riferimento alle proprietà proprie delle classi coinvolte o a quelle ereditate dagli antenati delle classi vincolata e vincolanti;
- **Autoanello**: se un vincolo coinvolge una classe sia come classe vincolata che come classe vincolante (autoanello), l'insieme di oggetti vincolanti applicati per soddisfare il vincolo su un certo oggetto O è costituito da tutti gli oggetti della classe ***escluso l'oggetto O stesso***;
- **Superfici con frontiera 3D**: se uno o entrambi gli attributi geometrici `f` e g sono di tipo GU_C*SurfaceB3D è necessario specificare la componente considerata nel vincolo, ossia l'attributo `B3D` o l'attributo `superficie`;
- **Abbreviazioni**: per semplificare la formulazione dei vincoli si può usare il codice alfanumerico della classe al posto del nome completo e inoltre il prefisso della classe può essere omesso davanti ai nomi degli attributi quando la classe è quella del contesto corrente;

- **Limiti nell’uso di attributi**: i vincoli non possono far riferimento agli attributi di un’associazione e agli attributi aggiuntivi di tipo base presenti nel dominio enumerato gerarchico di un attributo.

Inoltre, le regole per la rappresentazione grafica del vincolo esistenziale di base si applicano a tutti i vincoli e tutte le varianti:

1. il vincolo è rappresentato da una freccia che va dalla classe vincolata alla classe vincolante
2. sulla classe vincolata e vincolante, nei punti dove parte o arriva la freccia, sono indicate le componenti spaziali alle quali il vincolo fa riferimento (in questo punto si applicheranno anche le selezioni e le funzioni utilizzate dalle varianti descritte nel seguito)
3. la freccia è documentata da un rettangolo che contiene la relazione topologica (IN nell’esempio), il nome del vincolo (VT_elemento_Stradale nell’esempio) e il tipo generale di vincolo (tipo=esiste nell’esempio).

Definizione formale del vincolo esistenziale tramite regole di traduzione in OCL

La definizione formale di tutti i vincoli in appendice A è costituita dalle seguenti parti.

1. ***Nome del vincolo***
2. ***Definizione dei simboli***
3. ***Sintassi del vincolo in GeoUML***, che evidenzia alcune parole chiave fisse tipiche del tipo di vincolo (*vincolo* e *esiste* nel vincolo esistenziale di base) e alcune parti variabili, che definiscono a quali classi e componenti spaziali il vincolo si riferisce e quali relazioni topologiche applica
4. ***Template OCL corrispondente***: un template è una funzione che ha un nome (in questo caso `ExistentialTopoConstraint`) ed alcuni parametri (in questo caso `X, g, Y, f` e `DJ_R`). Tali parametri corrispondono alle parti variabili della formulazione del vincolo nella sintassi GeoUML e compaiono anche nell'espressione OCL che segue. Quando il vincolo viene utilizzato nella specifica tali parametri verranno sostituiti con il nome dei corrispondenti costrutti dello schema su cui si vuole far agire il vincolo.
   Nella formulazione del vincolo in OCL si fa uso dell'operazione `check` definita sull'oggetto GU_Object, che consente di verificare il soddisfacimento di una disgiunzione di relazioni topologiche $r_1,\ldots,r_n$ invece che di una sola relazione topologica; formalmente check, applicata a un oggetto *a* con riferimento a un oggetto *b*, è definita nel modo seguente:
   $a.check(\{r_1,\ldots,r_n\}, b) \equiv_{def} a.r_1(b) \vee \ldots \vee a.r_n(b)$

Nel caso del vincolo esistenziale di base la formalizzazione assume la forma seguente:.

Definizione dei simboli - *Date due classi X e Y contenenti almeno un attributo geometrico ciascuna, rispettivamente g e f, il vincolo topologico esistenziale da X verso Y, basato sulla disgiunzione di relazioni DJ_R = {rel$_1$,...,rel$_n$} si definisce come segue:*

Sintassi:

vincolo X.g (rel$_1$ | … | rel$_n$) *esiste* Y.f

Template OCL:

```
ExistentialTopoConstraint (X, g, Y, f, DJ_R) ≡
context X
inv: Y.allInstances ->
      exists (a:Y | self.g.check(DJ_R, a.f))
```

Esempio di applicazione della regola formale alla traduzione di un vincolo specifico

Applicando questa regola al vincolo dell'esempio precedente:

> *vincolo* ElementoStradale.Percorso (IN) *esiste* AreaStradale.Estensione

si vede che dobbiamo sostituire la lista di parametri `(X, g, Y, f, DJ_R)` con i seguenti (ElementoStradale, Percorso, AreaStradale, Estensione, IN) e otteniamo il seguente vincolo in OCL:

```
context ElementoStradale
inv: AreaStradale.allInstances ->
     exists(a: AreaStradale |
          self. Percorso.check(IN, a.Estensione))
```

Il vincolo in questa forma potrebbe essere inserito in un AS standard, a condizione di avere definito la funzione `check` sulla generica geometria.

Varianti del vincolo topologico esistenziale di base

Le varianti del vincolo topologico esistenziale definite nel seguito possono essere applicate anche in combinazione tra loro.

Vincolo topologico esistenziale con selezioni

Questa variante permette di selezionare gli oggetti delle classi coinvolte nel vincolo.

Commento e Esempio

Una selezione sulla classe vincolata limita gli oggetti che devono soddisfare il vincolo a quelli che soddisfano la selezione, rendendo il vincolo più debole; una selezione sulla classe vincolante riduce gli oggetti che possono essere utilizzati per soddisfare il vincolo, rendendolo più forte.
Nel seguente esempio si restringe il vincolo dell'esempio precedente richiedendo che solo gli *ElementoStradale* di un certo tipo debbano soddisfare il vincolo di essere contenuti in un oggetto di *Area Stradale* (la definizione della classe Elemento Stradale deve contenere anche l'attributo tipo):

vincolo (ElementoStradale.tipo="T1")ElementoStradale.percorso
(IN) *esiste* AreaStradale.estensione

Interpretazione del valore nullo nella valutazione dei vincoli

Il GeoUML ammette l'opzionalità dei valori degli attributi/ruoli di una classe e modella l'assenza di valore attraverso il concetto di `nullo`;
La definizione dei vincoli implica la possibile selezione di oggetti e/o tratti (eventi, sottoaree) e la possibile applicazione della funzione boundary che possono produrre un insieme vuoto di oggetti e/o di geometrie).
La semantica dei vincoli rispetto a questo problema è governata dalle seguenti regole:

- qualsiasi funzione (ad esempio, boundary()) applicata ad un valore nullo produce un valore nullo come risultato;
- l'interpretazione delle selezioni applicate alle classi (ai tratti,…) sottoposte ad un vincolo fa riferimento alla semantica standard dell'SQL92 anche per l'interpretazione del valore nullo; si ricorda che la presenza di un valore nullo durante la valutazione di una condizione può generare un risultato "unknown" (non è vero e non è falso); in tali casi l'SQL forza a falso la valutazione e non seleziona l'oggetto. La stessa condizione si verifica nella selezione dei tratti (eventi, sottoaree);
- le geometrie (della classe vincolata e di quelle vincolanti) concorrono alla valutazione di un vincolo solo se contengono un valore non nullo; un vincolo stabilisce pertanto una condizione che deve essere soddisfatta dalle sole geometrie realmente disponibili (anche vuote) all'atto della valutazione del vincolo.

Vincolo topologico esistenziale sulla frontiera o sulla proiezione planare

Questa variante permette di applicare le funzioni BND (`boundary()`) e/o PLN (`planar()`) alle componenti spaziali coinvolte nel vincolo. Le funzioni possono anche essere applicate in cascata.

Esempio

Si consideri l'esempio del paragrafo precedente con la seguente variante: si ipotizza che la componente spaziale Percorso della classe ElementoStradale abbia geometria in 3D; in questo caso è necessario applicare la funzione *planar()* nel vincolo per soddisfare la regola generale che le relazioni topologiche si applicano a geometrie nello stesso spazio:

vincolo (ElementoStradale.tipo="T1")ElementoStradale.percorso.*PLN*
(IN) *esiste* AreaStradale.estensione

La rappresentazione grafica dell'esempio è mostrata nel Diagramma 6.2. Si vede che nell'applicazione delle regole generali per la rappresentazione grafica la selezione e la funzione PLN sono indicate nel riquadro dedicato alla componente spaziale.

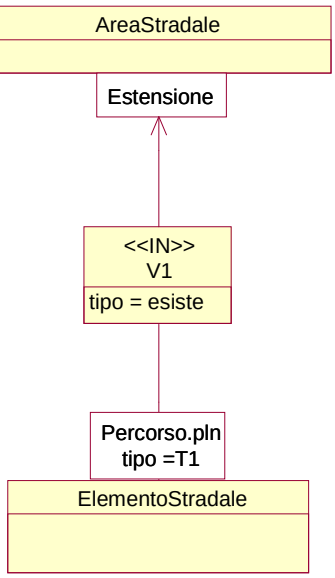


Diagramma 6.2- Esempio di vincolo topologico esistenziale sulla proiezione

Vincolo topologico collegato ad una associazione

Con questa variante si considerano, ai fini del soddisfacimento del vincolo, solo gli oggetti della classe vincolante che sono legati all'oggetto al'istanza della classe vincolata attraverso un'associazione specificata nello schema.

Esempio

La forma testuale di questa variante è illustrata dal seguente esempio, che utilizza un'associazione e la funzione `planar()`, per mostrare come le diverse varianti di vincoli possano combinarsi.

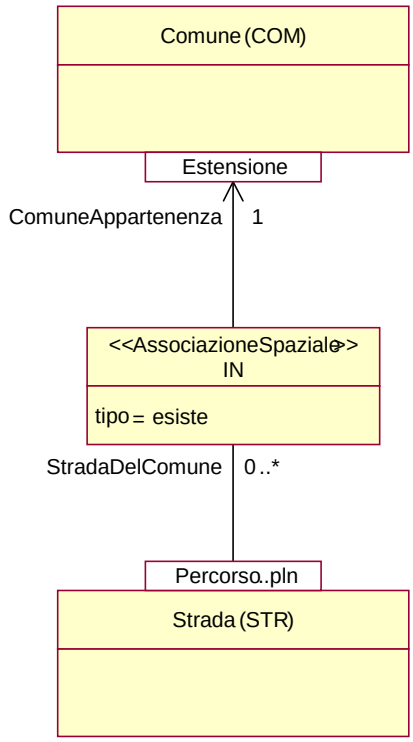
vincolo Strada.percorso.*PLN* (IN) *esiste*
Strada.comuneAppartenenza.estensione

tale vincolo opera su classi e componenti spaziali che devono essere definite nel modo seguente:

classe Strada (STR – 0501)
attributi
componenti spaziali della classe
.... percorso: GU_CPCurve3D
ruoli
ComuneAppartenenza [1..1] : Comune

classe Comune (COM - 0101)
attributi
componenti spaziali della classe
..... estensione: GU_CPSurface2D

La forma grafica estende le regole generali di rappresentazione, perchè fa coincidere la rappresentazione del vincolo con quella della associazione utilizzata, nel modo seguente:



Vincolo su attributi a tratti o a sottoaree

Il vincolo può anche fare riferimento alla geometria di un attributo a tratti, a tratti sul contorno, a eventi o a sottoaree; in tal caso occorre sostituire nella specifica del vincolo l'attributo geometrico con la chiamata di una delle funzioni che restituiscono i tratti, gli eventi o le sottoaree a seconda del tipo di attributo (ad esempio, le funzioni `TrattiDi_A()` e `SottoareeDi_A()`, dove `A` è il nome dell'attributo a tratti o a sottoaree); tali funzioni possono anche includere una selezione dei tratti o delle sottoaree considerate).
Nel seguito si farà riferimento all'attributo a tratti, tuttavia ogni vincolo potrà essere riformulato in modo simile per gli attributi a tratti sul contorno, a eventi e a sottoaree.

Sintassi:
Siano X e Y due classi e a e b due attributi a tratti appartenenti rispettivamente a X e Y; il riferimento agli attributi a tratti può avvenire su ambedue le classi, oppure solo sulla vincolata oppure solo sulla vincolante, nei modi seguenti

a. *vincolo* X.TrattiDi_a( ) (rel_1|…|rel_n) *esiste* Y.TrattiDi_b( )
   questa formulazione richiede che per ogni tratto dell'attributo a della classe X esista un tratto dell'attributo b della classe Y tale che tra i due valga la relazione specificata
b. *vincolo* X.TrattiDi_a() (rel_1 | … | rel_n) *esiste* Y.f
   questa formulazione richiede che per ogni tratto dell'attributo a della classe X esista un'istanza della classe Y con una componente spaziale f tale che tra i due valga la relazione specificata
c. *vincolo* X.g (rel_1 | … | rel_n) *esiste* Y.TrattiDi_b( )
   questa formulazione richiede che per la componente spaziale g di ogni istanza della classe X esista un tratto dell'attributo b della classe Y tale che tra i due valga la relazione specificata.

La sintassi grafica di queste varianti si ottiene dalla sintassi grafica del vincolo base sostituendo alla componente spaziale la funzione `TrattiDi_a()`, dove a è l'attributo a tratti coinvolto nel vincolo, nei rettangoli relativi alla componente spaziale.

Esempio
Il seguente vincolo segue la struttura del caso c, utilizza però le sottoaree invece dei tratti, applica una selezione sulla classe vincolata e una selezione sull'attributo a sottoaree della classe vincolante

(uso **=** "stradale") GALLER.Sup_sede.*superficie* **(CT)** *esiste*
AC_VEI.*SottoareeDi_*Sede(Sede **=** "in galleria")

Spiegazione.

1. si consideri un'istanza G della classe GALLER tale che il suo attributo uso="stradale" (selezione sula classe vincolata)
2. si consideri la sua componente spaziale Sup_sede, che è una superficie B3D, e si prenda la componente *superficie* di tale valore, che è nel piano 2D, e chiamiamo tale superficie GS
3. deve esistere una sottoarea SA dell'attributo a sottoaree Sede di una istanza della classe AC_VEI con valore "in galleria" tale che GS contiene SA.

Vincolo topologico su unione

Il vincolo topologico su unione fa riferimento all'unione (gUnion) delle componenti spaziali di tutte le istanze della classe vincolante Y, invece di richiedere l'esistenza di una singola istanza che soddisfi il vincolo.
In altri termini, la relazione topologica viene verificata rispetto alla geometria che si ottiene dall'unione dei valori delle componenti spaziali di tutte le istanze di *Y*.
Anche per il vincolo topologico su unione esistono le varianti presentate per il vincolo topologico esistenziale, vale a dire: la versione con selezione, la versione che si riferisce al boundary() e planar(), quella che si riferisce ad un'associazione e quella che si riferisce ai tratti.

Commento e Esempio

Un esempio della forma testuale del vincolo è il seguente:

vincolo ElementoStradale.percorso (IN) *unione* AreaStradale.estensione

tale vincolo opera su classi e componenti spaziali che devono essere definite nel modo seguente:

classe AreaStradale (...)
attributi
componenti spaziali della classe
....- estensione: GU_CPSurface2D
classe ElementoStradale (...)
attributi
componenti spaziali della classe
.... - percorso: GU_CPCurve2D

Questo vincolo richiede, per ogni ElementoStradale, che l'unione delle componenti spaziali "Estensione" di tutte le istanze della classe *AreaStradale* lo contenga: il vincolo è quindi molto più debole della versione esistenziale vista in precedenza.

La rappresentazione grafica del vincolo è ottenuta da quella del vincolo esistenziale sostituendo la dicitura tipo=unione alla dicitura tipo=esiste.

Vincolo topologico universale

Il vincolo topologico universale richiede che la relazione topologica sia presente tra l'oggetto vincolato e tutte le istanze della classe vincolante.
Anche il vincolo topologico universale può far riferimento ai singoli componenti di una superficie con contorno in 3D.
Inoltre esistono le varianti presentate per il vincolo topologico esistenziale, vale a dire: la versione con selezione, con le funzioni boundary e planar, con i tratti e quella che si riferisce ad un'associazione.

Commento e Esempio

Il vincolo topologico universale è usato quasi esclusivamente con la relazione spaziale Disjoint, eventualmente in disgiunzione con la relazione spaziale Touch. Ciò accade nel seguente esempio, che illustra anche la possibilità, valida per tutti i tipi di vincoli, che la classe vincolata e vincolante coincidano.

vincolo ElementoStradale.percorso (DJ | TC)
perOgni ElementoStradale.percorso

tale vincolo opera su classi e componenti spaziali che devono essere definite nel modo seguente:

classe ElementoStradale (ELESTR – 0504)
attributi
...
componenti spaziali della classe
...- percorso: GU_CPCurve3D

La rappresentazione grafica del vincolo è ottenuta da quella del vincolo esistenziale sostituendo la dicitura tipo=perOgni alla dicitura tipo=esiste.

Questo esempio richiama la regola generale relativa agli autoanelli nei vincoli: quando la classe vincolata coincide con la classe vincolante nella valutazione di un'istanza O della classe vincolata si deve fare riferimento alle istanze della stessa classe, intesa come vincolante, ma escludendo l'istanza O stessa, perché l'unica relazione spaziale di un oggetto con se stesso è *Equals*. Nell'esempio precedente ciò significa che una istanza di ElementoStradale deve essere in relazione DJ|TC con tutte le altre istanze della stessa classe (ma non ovviamente con se stessa).

Vincoli topologici con più classi vincolanti

Tutti i vincoli topologici possono fare riferimento a più classi vincolanti, ma è necessario fare attenzione alla interpretazione del loro significato.
La sintassi della forma di base in questo caso è la seguente:

Sintassi:

vincolo X.g… (rel$_1$ |…| rel$_n$) <*tipo*> (Y$_1$.f$_1$…, …, Y$_n$.f$_n$…)

dove tipo può essere:

esiste oppure *perOgni* oppure *unione*

Le varianti (selezioni, funzioni BND e PLN e riferimento a attributi a tratti o sottoaree) possono essere applicate separatamente alle singole classi vincolanti.

Per quanto riguarda il significato è indispensabile fare attenzione al diverso comportamento dei quantificatori:

1. nel caso esistenziale (*esiste*) si richiede che esista un elemento in una delle classi vincolanti, per cui il vincolo può essere soddisfatto da una istanza di *una qualsiasi* delle classi vincolanti
2. nel caso universale (*perOgni*) il vincolo deve essere soddisfatto da *tutte* le istanze selezionate di tutte le classi vincolanti
3. nel caso del vincolo su unione il vincolo deve essere soddisfatto dall'unione delle istanze delle classi vincolanti; in questo caso le geometrie delle classi vincolanti sono unite prima di effettuare la verifica del vincolo

Disgiunzione di vincoli topologici

La disgiunzione di vincoli topologici permette di indicare che per ogni elemento di una classe vincolata deve essere soddisfatto almeno uno dei vincoli posti nella disgiunzione.
Nella sintassi testuale i vincoli in disgiunzione sono separati dalla parola chiave OR e sono preceduti dalla parola chiave *disgiunzione*.

Commento e Esempio

E' necessario fare attenzione che *tutti i vincoli della stessa disgiunzione facciano riferimento alla stessa classe vincolata*.
Un esempio di disgiunzione di vincoli topologici in forma testuale è il seguente. Esso richiede che un *ElementoStradale* sia disgiunto da ogni altro *ElementoStradale* oppure la sua frontiera appartenga all'insieme delle *Giunzioni*:

disgiunzione ElementoStradale.percorso (DJ) *perOgni* ElementoStradale.percorso
OR
ElementoStradale.percorso.*BND* (IN) *unione* Giunzione.luogo

Dal punto di vista grafico una disgiunzione di vincoli topologici si rappresenta congiungendo con una linea i vincoli che sono in disgiunzione, come nel Diagramma 6.6.

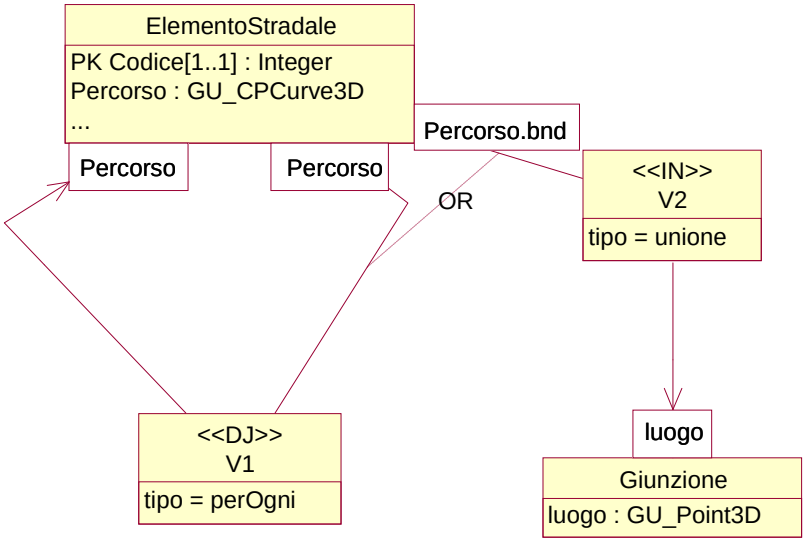


Diagramma 6.6 - Esempio di disgiunzione di vincoli topologici

Vincoli di composizione (vincoli part_whole)

Questa categoria di vincoli è costituita da un vincolo fondamentale, il vincolo di composizione (*compostoDa*), e da un vincolo derivato, il vincolo di partizione.
Il vincolo di partizione è derivato in quanto esprimibile tramite il vincolo di composizione e alcuni vincoli topologici leggermente modificati. Dato che la combinazione di vincoli topologici modificati ha una sua utilità specifica, viene definito un vincolo ad hoc, detto di appartenenza, per esprimere tale combinazione.

In sintesi, i vincoli di composizione sono i seguenti:

1. il vincolo di composizione, che è fondamentale e non derivabile dai vincoli topologici
2. i vincoli di appartenenza disgiunta (dj_IN e qdj_IN),
3. il vincolo di partizione, che sarebbe esprimibile tramite un vincolo di composizione e un vincolo di appartenenza.

Si noti che anche i vincoli di composizione possono essere coinvolti in una espressione di disgiunzione di vincoli.

Anche per i vincoli di composizione sono applicabili alcune delle varianti già introdotte per i vincoli topologici, in particolare è possibile aggiungere selezioni e riferire il vincolo alla frontiera o alla proiezione planare del valore geometrico. Infine è possibile riferire il vincolo alla geometria di un attributo a tratti, a eventi o a sottoaree e legarli ad associazioni.

Vincolo di composizione

Il vincolo di composizione definisce un vincolo tra un attributo geometrico *f* di una classe *Y* e l'attributo geometrico *g* di una classe *X*. Tale vincolo stabilisce che per ogni oggetto y di *Y* l'attributo *f* sia uguale all'unione degli attributi geometrici *g* di uno o più oggetti x di *X;* nel caso in cui il vincolo sia collegato ad un'associazione il vincolo è più stringente, perché richiede che per ogni oggetto y di Y l'attributo *f* sia uguale all'unione degli attributi geometrici *g* di tutti gli oggetti di X collegati a y nell'associazione. Si noti che in tutti i casi gli oggetti x di X che contribuiscono a soddisfare il vincolo relativamente a un oggetto y di Y hanno una geometria contenuta nella (o uguale alla) geometria di y.
Il vincolo di composizione è di natura esistenziale: infatti esso richiede che, dato l'attributo geometrico *f* di un oggetto della classe vincolata (classe composta), esistano nella classe vincolante (classe componente) un numero intero di istanze le cui componenti spaziali, in unione geometrica tra loro, siano uguali a f.
Sintassi:

```
vincolo Y.f compostoDa X.g
```

La rappresentazione grafica del vincolo è ottenuta da quella del vincolo esistenziale sostituendo la dicitura "compostoDa" al posto della relazione spaziale e dell'indicazione del tipo di vincolo.
Il vincolo di composizione impone che le geometrie delle classi vincolata e vincolanti siano tutte della stessa dimensione (ad esempio, solo curve o solo superfici) perché non è possibile comporre un oggetto di dimensione diversa da quella dei componenti (ad esempio, una superficie non può essere ottenuta tramite l'unione di curve).

Esempio

I seguenti 2 esempi fanno riferimento alla classe vincolata EstesaAmministrativa e utilizzano il vincolo di composizione sia sulla componente spaziale areale (Pertinenza), sia sulla componente spaziale lineare (Tracciato_analitico):

1. ES_AMM.Pertinenza.*superficie* *compostoDa* AR_STR.Estensione.*superficie*
   - questo vincolo richiede che esistano un certo numero di aree stradali (AR_STR) che compongono la Pertinenza di una estesa amministrativa;
   - il vincolo impone indirettamente che le istanze di AR_STR siano tagliate dove termina un'estesa amministrativa;
   - si noti la necessità di riferirsi all'attributo *superficie* perchè sia Pertinenza che Estensione sono superfici con frontiera 3D.
2. ES_AMM.Tracciato_analitico *compostoDa* EL_STR.Tracciato
   - questo vincolo richiede che esistano un certo numero di elementi stradali (EL_STR) che compongono il Tracciato di una estesa amministrativa;
   - il vincolo impone indirettamente che le istanze di EL_STR siano tagliate dove termina un'estesa amministrativa.

tale vincolo opera su classi e componenti spaziali che devono essere definite nel modo seguente:

classe ... (Es_AMM ...)
attributi
componenti spaziali della classe
... - Pertinenza: GU_CPSurfaceB3D
... – Tracciato: GU_CPCurve3D
classe .... (AR_STR ...)
attributi

componenti spaziali della classe
... - Estensione: GU_CPSurfaceB3D
classe .... (EL_STR ...)
attributi
componenti spaziali della classe
... - Tracciato: GU_CPCurve3D

Il vincolo di appartenenza

Il vincolo di appartenenza (contenimento geometrico) della componente spaziale *g* di una istanza della classe vincolata *X* alla componente spaziale *f* di una istanza della classe vincolante *Y* si ottiene semplicemente usando un vincolo topologico esistenziale dove si richiede la relazione topologica *IN* tra *g* e *f*.
Non sarebbe necessario definire quindi nessun vincolo specifico per esprimere tale proprietà. Tuttavia, spesso interessa anche indicare quali relazioni topologiche sono ammesse tra le istanze della classe vincolata che appartengono alla stessa istanza della classe vincolante.

Per soddisfare tale esigenza si introducono i seguenti due vincoli, che combinano l'appartenenza (IN) con la disgiunzione degli elementi in appartenenza

- il vincolo di appartenenza con disgiunzione (*dj-IN*): tale vincolo richiede che ogni istanza della classe vincolata sia contenuta geometricamente nella classe vincolante (vincolo topologico IN) e che sussista, tra le istanze della classe vincolata che sono contenute nella stessa istanza di classe vincolante, la relazione *Disjoint* oppure di *Touch* ristretta al caso in cui esista solo l'intersezione di frontiera con frontiera (nota: la relazione di *Touch* ammette anche l'intersezione di frontiera e parte interna);
- il vincolo di appartenenza quasi-disgiunta (*qdj-IN*): tale vincolo vale solo per oggetti geometrici di tipo GU_C*Curve*D (incluse le specializzazioni) e consente che tra le istanze della classe vincolata sussista oltre alle relazione *Disjoint* e *Touch* anche la relazione *Cross*.

La rappresentazione grafica si ottiene da quella del vincolo di composizione sostituendo la dicitura <<compostoDa>> con <<dj-IN>> o <<qdj-IN>>.

Esempio
Il seguente vincolo dice che il percorso di un reticolo idrografico naturale (RT_IDN) deve appartenere ad un reticolo idrografico (RT_IDR) e che tutti i reticoli naturali appartenenti ad uno stesso reticolo idrografico devono essere disgiunti oppure in Touch frontiera su frontiera tra loro.

RT_IDN.Percorso *dj-IN* RT_IDR.Sviluppo

tale vincolo opera su classi e componenti spaziali che devono essere definite nel modo seguente:

classe ... (RT_IDN ...)
attributi
componenti spaziali della classe
... - Percorso: GU_CXCurve3D
classe ...(RT_IDR ...)
attributi
componenti spaziali della classe
... – Sviluppo: GU_CXCurve3D

Anche per i vincoli di appartenenza sono applicabili alcune delle varianti già introdotte per i vincoli topologici, in particolare è possibile aggiungere selezioni, riferire il vincolo

alla frontiera o alla proiezione planare del valore geometrico o alla geometria di un attributo a tratti, a eventi o a sottoaree o ad un’associazione.

Il vincolo di partizione

Il vincolo di partizione (*partizionato)* esprime la combinazione di un vincolo di composizione con un vincolo di appartenenza disgiunta (nelle due versioni *dj* e *qdj*).
Il vincolo di partizione è espresso nel modo seguente.
Sintassi:

vincolo Y.f *partizionato* X.g
oppure
vincolo Y.f q-*partizionato* X.g

con il seguente significato:

- L'unione degli attributi geometrici *g* degli oggetti della classe *X* che partizionano un oggetto della classe *Y* compone l'attributo geometrico *f* dell'oggetto della classe *Y*. (cioè vale il vincolo Y.f *compostoDa* X.g);
- Gli attributi geometrici *g* degli oggetti della classe *X* che formano la partizione di *f* non si sovrappongono (al più sono adiacenti), cioè vale il vincolo X.g *dj-IN* Y.f (oppure X.g q*dj-IN* Y.f).

La rappresentazione grafica si ottiene da quella del vincolo di composizione sostituendo la dicitura <<compostoDa>> con <<partizionato>> o <<q-partizionato>>.

Esempio

Si considerino due classi relative alle Regioni ed alle Province legate da un'associazione Regioni-Province che è indipendente dalla rappresentazione geometrica dei loro territori. Se realizziamo tale rappresentazione geometrica in forma di poligoni, allora è evidente che esiste un vincolo di partizione tra il poligono di una regione e quelli delle sue province, che potremmo definire con il vincolo *partizionato* nel modo seguente:

vincolo Regione.estensione *partizionato*
Regione.ProvinceDellaRegione.estensione

tale vincolo opera su classi e componenti spaziali che devono essere definite nel modo seguente:

classe Regione (...)
attributi
componenti spaziali della classe
... - Estensione: GU_CPSurface2D;
ruoli: ProvinceDellaRegione [1..*]: Provincia
...
classe Provincia (...)
attributi:
componenti spaziali della classe
.... - Estensione: GU_CPSurface2D;

Questo esempio mostra la combinazione del vincolo di partizione con il riferimento ad un'associazione, applicando la regola generale che le varianti introdotte per il vincolo esistenziale possono essere applicate anche ai vincoli di composizione.

Vincoli di composizione con più classi vincolanti

Esiste una variante dei vincoli *compostoDa*, *partizionato* oppure *q-partizionato* per permettere che facciano riferimento all'unione dei valori di attributi geometrici di diverse classi vincolanti. Le diverse classi con i rispettivi attributi devono in questo caso essere elencate tra parentesi come mostrato nel seguente esempio.

Commento e Esempio

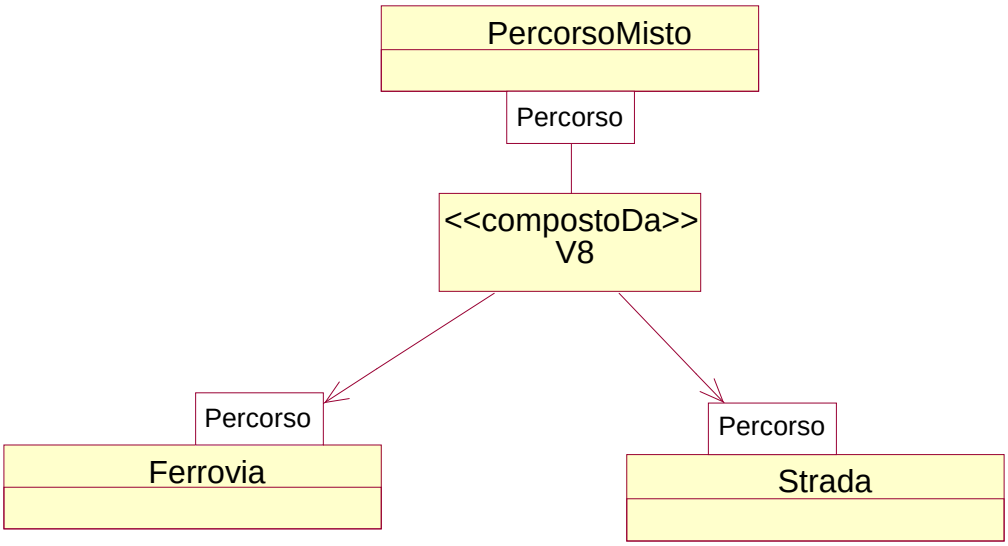
Si vuole definire un "percorso misto" come costituito da tratti stradali e tratti ferroviari, tale vincolo è esprimibile solo precisando come classi vincolanti sia la classe che rappresenta i tratti stradali sia quella che rappresenta i tratti ferroviari, come nell'esempio seguente.

vincolo PercorsoMisto.percorso *compostoDa*
(Strada.percorso, Ferrovia.percorso)

tale vincolo opera su classi e componenti spaziali che devono essere definite nel modo seguente:

classe Strada (STR – 0501)
attributi
componenti spaziali della classe
... - percorso: GU_CPCurve2D
classe Ferrovia (FER – 0601)
attributi
componenti spaziali della classe
... - percorso: GU_CPCurve2D
classe PercorsoMisto (PMS - 0701)
attributi
componenti spaziali della classe
... - percorso: GU_CPCurve2D

Il significato di questa dichiarazione è che un oggetto di tipo "percorso misto" possiede un attributo geometrico costituito dall'unione di curve che appartengono agli attributi geometrici delle strade o delle ferrovie.
La forma grafica è mostrata nel diagramma seguente, dove la freccia che parte dalla classe vincolata si divide per arrivare a tutte le classi vincolanti.



Gestione delle superfici collassate

Le componenti geometriche di tipo GU_C*Surface* di alcune classi possono essere collassabili. Ciò significa che si ammette la possibilità che alcune (o tutte le) istanze di una classe abbiano come geometria di un attributo di tipo GU_C*Surface* non solo una superficie, ma una curva, un punto o una combinazione di punti curve e superficie.
La causa del collassamento è legata alle dimensioni dell'oggetto in funzione dell'accuratezza metrica prevista per la scala di rilievo.
Nello Schema Concettuale si indicano le componenti spaziali delle classi che ammettono il collassamento, in modo da introdurre a livello di modello implementativo meccanismi particolari per la gestione delle geometrie collassate solo quando effettivamente servono. Il modo di indicare nella specifica la possibilità di collassamento è descritto nel capitolo successivo.

Questa scelta consente di non alterare la specifica concettuale di una classe in presenza di collassamento di una sua componente geometrica (una superficie che collassa rimane comunque una superficie) e di gestire a livello di Modello Implementativo la rappresentazione e il corretto trattamento delle geometrie collassate.

Inoltre si precisa che:

- Gli attributi a sottoaree su superfici collassate non assumono valore sulla parte di geometria che collassa a curve o punti. Ovviamente se il collassamento è completo, gli attributi a sottoaree non vengono rappresentati.
- Gli attributi a tratti sul contorno di superfici collassate non assumono valore sulla parte collassata.

In questo capitolo si definisce a livello del modello concettuale quali forme può assumere la geometria di una superficie collassata e si definisce il suo comportamento quando questa partecipa ai vincoli.

Proprietà e valori ammessi

Ogni superficie collassata **Sc** di tipo GU_CPSurface2D, GU_CXSurface2D, GU_CPSurfaceB3D e GU_CXSurfaceB3D viene rappresentata da tre componenti:

- la porzione degenerata a curva **Sc.curve** di tipo GU_CXCurve2D,
- la porzione degenerata a punto **Sc.point** di tipo GU_CXPoint2D e
- la porzione non degenerata **Sc.surface** di tipo GU_CXSurface2D.

L'insieme delle 3 componenti eterogenee nel tipo non viene assimilato ad un aggregato generico (privo di frontiera, parte interna e relazioni topologiche) e preserva le seguenti proprietà dei rispettivi tipi originali:

- di essere un oggetto regolare, pertanto i buchi interni alla superficie non possono mai collassare a curve o punti (tali collassamenti provocano l'eliminazione del buco), altrimenti si creerebbero dei tagli o punture non ammessi dalla proprietà;
- dimension(), isCycle() e IsSimple() conservano i valori della superficie originale anche qualora la superficie collassata sia composta solo da curve e/o punti;
- CoordinateDimension() dei componenti Sc.curve e Sc.point coincidono con quello della componente Sc.surface.

La frontiera e la parte interna di queste superfici vengono ridefinite attraverso una reinterpretazione delle componenti di tipo curva e punto. Considerando l'insieme dei punti che descrive una superficie non collassata si può interpretare il collassamento come una funzione suriettiva tra la superficie non collassata e quella collassata come mostrato dal collassamento della superficie di Figura 7.1.a nella superficie della Figura 7.1.b dove molti punti diversi della superficie non collassata convergono allo stesso

punto in quella collassata.

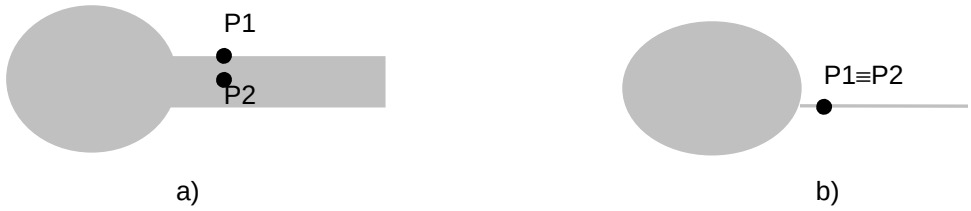


Figura 7.1

Ciò significa che la parte interna e la frontiera nella porzione di superficie collassata diventano indistinguibili e quindi le componenti di tipo curva e punto sono da interpretare come appartenenti contemporaneamente alla frontiera e alla porzione interna della superficie collassata complessiva. Pertanto se Sc è una superficie collassata la funzione boundary() viene così ridefinita:

Sc.boundary() = Sc.point.gUnion(Sc.surface.boundary().gUnion(Sc.curve))

e la sua parte interna I(Sc) non è più la differenza Sc.PS() – Sc.boundary().PS() ma diventa:

I(Sc) = Sc.curve.gUnion(Sc.point).PS() ∪ I(Sc.surface**)**,

quindi la parte interna non è disgiunta dalla frontiera.

Anche le operazioni insiemistiche richiedono una ridefinizione. Ad esempio, l'operazione insiemistica gUnion applicata a due superfici collassate Sc1 e Sc2 produce in generale una superficie collassata le cui componenti si ottengono nel seguente modo:

- Si esegue l'unione insiemistica di tutte le componenti di Sc1 e di Sc2,
- Si estraggono dall'insieme risultato le tre componenti separatamente.

Si noti che le curve e i punti che intersecano superfici vengono assorbite dalle medesime per effetto dell'unione insiemistica.

Nel seguito si definiscono
Precisano, per ognuno dei tipi che rappresentano superfici in GeoUML, le proprietà aggiuntive che devono soddisfare le tre componenti e che unitamente alle precedenti permettono di identificare quali siano i valori ammessi per una superficie collassata:

GU_CPSurface2D

Proprietà 1 (si esclude la degenerazione parziale a punti): **Sc.point** può essere diverso dal valore nullo solo se sia **Sc.curve** che **Sc.surface** sono nulli e in tal caso essa può essere composta da un solo punto. Questa proprietà permette il mantenimento della connessione della superficie.
Proprietà 2 (si esclude la degenerazione a insieme non connesso): per ogni coppia di punti di S= **Sc.curve**.gUnion(**Sc.surface**) deve esistere una curva di tipo GU_CPCurve* che li collega ed è completamente contenuta in S; la proprietà di connessione di una superficie collassata è quindi più debole di quella definita per una superficie non collassata perché considera anche i punti di frontiera nella valutazione.
Proprietà 3 (si esclude la sovrapposzione lineare tra le componenti): la relazione topologica tra la componente curva e la frontiera della componente superficie deve essere un Touch: **Sc.curve** TC **Sc.surface.boundary().**

GU_CXSurface2D

Per le superfici di tipo CX non è richiesta la connessione, mentre è ammessa la degenerazione parziale a punti, quindi rimane solo l'ultima proprietà così ridefinita:

Proprietà 1 (si esclude il self-overlapping tra le componenti): le relazioni ammesse tra le componenti di Sc sono le seguenti:

Disjoint o Touch tra curva e superficie: **Sc.curve** DJ **Sc.surface** OR **Sc.curve TC Sc.surface.boundary()**.
Disjoint tra punto e curva: **Sc.point** DJ **Sc.curve**
Disjoint tra punto e superficie: **Sc.point** DJ **Sc.surface**

GU_CP(CX)SurfaceB3D

Per le superfici di tipo B3D la porzione superficie degenera come definito in precedenza. In aggiunta occorre gestire il collassamento dell'anello 3D che deve essere rappresentato da due componenti

- una componente detta Sc.B3D.curve di tipo GU_CXCurve3D (che non costituisce più in generale un anello) e
- una componente detta Sc.B3D.point di tipo GU_CXPoint3D, che rappresenta la parte del contorno collassata a punti.

Le proprietà che questo insieme di valori deve soddisfare è il seguente:
Il vincolo di uguaglianza del boundary della superficie 2D con la proiezione della B3D deve essere ridefinito come segue:

Sc.B3D.curve.gUnion(**Sc.B3D.point**).planar() =
Sc.point.gUnion(**Sc.surface**.boundary().gUnion(**Sc.curve**))

La figura 7.2 mostra alcuni casi di collassamento ammessi.

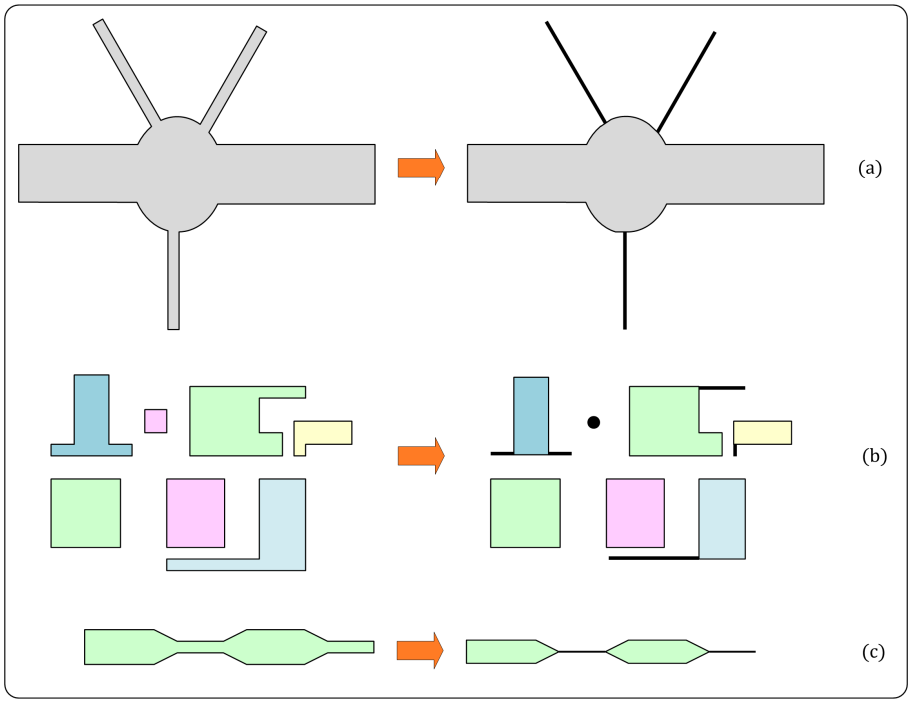


Figura 7.2: Alcuni esempi di collassamenti ammessi.

Relazioni e vincoli topologici

Le relazioni topologiche che coinvolgono le superfici collassate e i vincoli topologici basati su di esse saranno valutati considerando esclusivamente la componente poligonale delle superfici collassate Sc.surface e ignorando le componenti collassate SC.curve e Sc.point.
Pertanto è possibile che alcuni vincoli risultino violati a causa del collassamento.
Una sperimentazione dell'uso di tali superfici potrà permettere di definire delle linee guida per la creazione e il mantenimento delle superfici collassate e quindi per una valutazione più adeguata dei vincoli che le coinvolgono.

Commenti

Il collassamento di una superficie modifica la sua forma provocandone un restringimento che in generale produce la violazione delle relazioni topologiche (diverse da DJ) che la superficie soddisfaceva con altre geometrie prima del collassamento (ad es., la superifice *S* di Figura 7.3.a quando collassa nella superficie *Sc* di Figura 7.3..b trasforma la relazione TC che ha con la curva *c* nella relazione DJ).

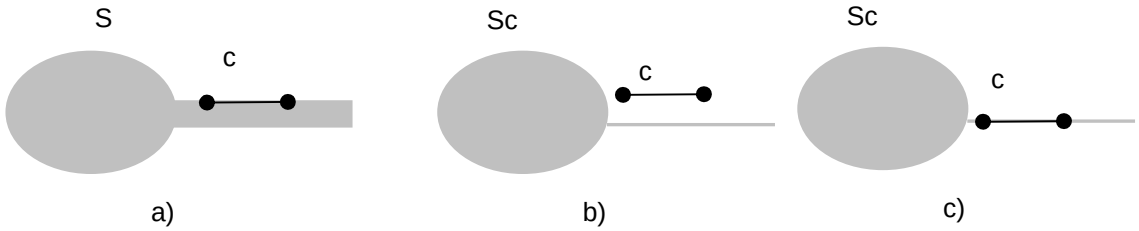


Figura 7.3

Conservare la relazione topologica dopo il collassamento può richiedere in generale di modificare la posizione delle geometrie in relazione topologica con la superficie (ad esempio, la curva *c* di Figura 7.3.a viene spostata sulla frontiera della componente collassata Sc.curve in Figura 7.3.c, per conservare la relazione TC) oppure di modificare la stessa forma delle geometrie coinvolte (ad esempio, il collassamento di una superficie in adiacenza con un'altra superficie comporta la dilatazione di quest'ultima per mantenere la relazione con la superficie collassata).
La modifica delle geometrie coinvolte dal collassamento di una superficie può produrre tuttavia una reazione a catena poiché tale modifica può a sua volta provocare la violazione delle relazioni topologiche che queste geometrie avevano nei confronti di ulteriori geometrie rendendo in generale difficile la conservazione delle relazioni topologiche complessive. Le relazioni topologiche sono usate nella formulazione dei vincoli topologici e in alcuni casi tali vincoli possono essere soddisfatti senza conservare le relazioni topologiche tra le geometrie (ad esempio, in un vincolo "DJ or TC" è possibile non alterare la geometria di un oggetto inizialmente in TC con una superficie che collassando soddisfa comunque la relazione DJ con l'oggetto). Tuttavia, in generale anche i vincoli richiedono di modificare le geometrie (ad esempio, un vincolo di partizione per la copertura del suolo impone che il collassamento di una superficie componente sia compensato dalle geometrie delle superfici adiacenti).
Inoltre si fa notare che, anche qualora fosse possibile modificare le geometrie coinvolte da un collassamento preservando le relazioni topologiche, sarebbe necessaria una ridefinizione delle stesse relazioni topologiche (ad esempio, lo spostamento della curva *c* dalla frontiera della superficie S di Figura 7.3.a alla frontiera della componente Sc.curve di Figura 7.3.c corrisponde a trasformare la relazione TC di Figura 7.3.a nella relazione IN tra la curva *c* e la componente *g.curve* considerata come curva di Figura 7.3.c).

Infine, poiché il collassamento di una superficie coinvolge più geometrie è possibile che geometrie che rispettano relazioni topologiche diverse con la superficie prima del collassamento possano convergere alla stessa relazione topologica dopo il collassamento (ad esempio, le due curve *c1* e *c2* di Figura 7.4.a che soddisfano le relazioni TC e IN rispettivamente con la superficie si ritrovano a soddisfare la stessa relazione topologica dopo il collassamento come mostrato in Figura 7.4.b). Ciò significa che la valutazione delle relazioni topologiche su una superficie collassata deve tener conto di quali siano le relazioni topologiche potenziali che sarebbero soddisfatte dalla superficie non collassata (ad esempio, la superficie *Sc* di Figura 7.4.b soddisfa quindi entrambe le relazioni CT e TC con entrambe le curve *c1* e *c2*); da ciò si deduce che le relazioni topologiche sulla superficie collassata non preservano più la proprietà di essere mutuamente esclusive tra di loro.

a) b)

Figura 7.4

Popolamento alle diverse scale e conformità

Definizione del popolamento ai diversi livelli di scala

E' possibile differenziare il popolamento delle classi e degli altri costrutti in base ai livelli di scala. Un livello di scala consiste in un insieme di scale indifferenziate dal punto di vista del popolamento. In una specifica è possibile definire un certo numero di livelli di scala e le scale rappresentate da ognuno di tali livelli.
I valori di popolamento indicano, per ognuno dei livelli di scala, se il costrutto deve essere popolato, cioè se deve contenere dei valori.
In una specifica differenziata sono definiti i valori di popolamento ai diversi livelli di scala per ognuno dei seguenti costrutti:

- ogni classe,
- ogni attributo,
- ogni valore di dominio enumerato,
- ogni componente spaziale,
- ogni attributo a tratti o sottoaree.

Il significato del popolamento è diverso per le classi e per gli altri costrutti.

Livelli di scala del National Core
Il National Core è stato definito, in base a motivazioni che non vengono riportate qui, facendo riferimento a due livelli di scala di popolamento: il livello 1000-2000 e il livello 5000-10000, indicati con NC1 per le scale 1000 e 2000 e NC5 per le scale 5000 e 10000.

Popolamento delle classi
Per quanto riguarda le classi, il popolamento di una classe C a un certo livello di scala LS significa che in un rilievo a una scala S appartenente a LS devono essere rilevate le istanze della classe C presenti nel mondo reale in funzione dell'accuratezza metrica prevista per la scala S.
Si noti che in taluni casi l'insieme di tali istanze può essere vuoto, tuttavia una classe popolata con zero istanze (classe vuota) rappresenta un contenuto informativo diverso da una classe non popolata.
Esempio:
Si consideri la classe ACQ_TER (acque territoriali). E' evidente che in un Data Product di una regione che non si affaccia sul mare tale classe non possiede istanze anche se venisse definita popolata, tuttavia il contenuto informativo della classe vuota dichiara che tale regione non possiede acque territoriali, mentre se la classe è non popolata il contenuto informativo non dice nulla relativamente a questo aspetto.
Il non popolamento ad un livello di scala di una classe che non è superclasse in una gerarchia implica pertanto che non abbia senso definire il popolamento dei suoi attributi, i domini embedded a quel livello di scala.
La caratteristica di popolamento non ha senso per le superclassi astratte poiché per definizione esse non possono avere mai istanze dirette.
Le superclassi astratte oppure non popolate ad uno o più livelli di scala devono invece definire il popolamento dei propri attributi e relativi domini; in tal modo le proprietà di popolamento sono ereditate dalle sottoclassi dirette o indirette della superclasse assieme alle altre caratteristiche (attributi, domini, associazioni, vincoli).

Popolamento degli altri costrutti

I costrutti diversi dalle classi non sono indipendenti, ma esistono esclusivamente all'interno delle classi. Essi rappresentano le proprietà delle classi. Se una di queste proprietà, ad esempio la proprietà P di una classe C, è popolata a un livello di scala LS, significa che nella struttura delle istanze della classe C rilevate alla scala S (appartenente a LS) tale proprietà è prevista, altrimenti no. La conseguenza di questa interpretazione è la seguente: ***se una classe C, popolata a due (o più) livelli di scala LS1 e LS2, possiede delle proprietà che sono popolate diversamente ai due livelli LS1 ed LS2, allora le istanze della classe C avranno una diversa struttura di proprietà ai due livelli di scala.***

Analogamente a quanto detto per il popolamento delle classi, anche per le proprietà di una classe il concetto di non-popolamento è diverso da quello di valore nullo applicabile a proprietà opzionali; se una proprietà P di una classe C non è popolata ad un livello di scala LS questo significa che non si sa nulla relativamente al valore di P in tutte le istanze di C rilevate ad una scala S di LS; invece il fatto che P sia popolata ma abbia un valore nullo in una certa istanza ha un preciso significato.

Il popolamento di una classe ad un livello di scala LS1 impone i seguenti vincoli:

- almeno un attributo proprio della classe o ereditato deve essere popolato in LS1;
- un attributo geometrico non popolato in LS1 implica il non popolamento degli attributi di attributo geometrico, attributi a tratti, eventi e a sottoaree definiti su di esso.
- il dominio embedded di ogni attributo popolato in LS1 deve avere almeno un valore popolato in LS1 e il dominio embedded di ogni attributo non popolato in LS1 non deve avere alcun valore popolato in LS1;

Determinazione della Scala di Rilievo

In una operazione di rilievo l'interpretazione dei valori di popolamento deve essere supportata da un insieme di **Regole di Determinazione della Scala di Rilievo** che stabiliscono a quale scala si deve rilevare un certo oggetto del territorio.

Esempi di possibili Regole di Determinazione della Scala di Rilievo sono:

- predefinire le caratteristiche del territorio che ne determinano la scala di rilievo, ad esempio urbanizzato a scala 1000-2000 e non urbanizzato a 5000-10000, oppure
- delimitare a priori le aree di rilievo omogenee alle varie scale
- associare la scala di rilievo a singole classi di oggetti da rilevare, ad esempio le strade al 1000, l'uso del suolo al 10000, ecc...

Le **Specifiche di Contenuto non stabiliscono Regole di Determinazione della Scala di Rilievo**, lasciando che siano appunto i capitolati di rilevamento a farlo.

Tuttavia una Specifiche di contenuto differenziata implica il soddisfacimento di alcune condizioni minime, illustrate nel seguito, da parte di un Data Product per essere conforme.

Classi normali e classi con istanze monoscala

La componente spaziale di una istanza di una classe normale può essere rilevata a diversi livelli di scala (tra quelli ai quali la classe è popolata). Pertanto, nelle classi normali non è possibile parlare della scala di rilievo di una singola istanza. La maggior parte delle classi appartiene a questa categoria.

Ad esempio, l'attributo spaziale "Pertinenza" di una istanza della classe "Estesa Amministrativa" si estende in modo da essere rilevato tipicamente a più scale diverse,

e non è possibile quindi definire un valore di scala di rilievo per ogni singola istanza.

Definiamo invece **classi a istanze monoscala** quelle classi per le quali la componente spaziale di ogni istanza deve essere rilevata tutta alla stessa scala (se la classe possiede più componenti spaziali, devono essere tutte rilevate alla stessa scala in una singola istanza). Queste classi sono individuate esplicitamente nella specifica di contenuto.
Per ogni istanza delle classi a istanze monoscala è evidentemente possibile definire una precisa Scala di Rilievo e quindi memorizzare tale informazione in un apposito attributo, che costituisce un metadato di istanza, chiamato **ScRil**, il cui dominio è un enumerato **LivScala** contenente i valori di livello di scala definiti nella specifica stessa.

L'esistenza di classi a istanze monoscala impone evidentemente dei vincoli alle Regole di Determinazione della Scala di Rilievo adottabili dal capitolato di rilievo: ad esempio, se si definiscono delle aree di rilievo a scala omogenea, le istanze delle classi a istanze monoscala non possono essere a cavallo di due diverse aree di rilievo.

Classi con Specifica omogenea o differenziata

In base ai valori di popolamento le classi presenti in una Specifica differenziata possono essere suddivise in due categorie:

1. Classi a Specifica Omogenea – sono le classi per le quali tutti i costrutti citati al precedente punto 8.1 sono popolati o non popolati in maniera identica ai diversi livelli di scala
2. Classi a Specifica Differenziata – sono tutte le altre classi

Tra la suddivisione delle classi in base all'omogeneità del popolamento e quella trattata precedentemente, in base alla possibilità delle istanze di avere componenti spaziali estese su più scale, esiste solamente il seguente vincolo:

> **Tutte le classi a Specifica Differenziata devono essere Classi a Istanze Monoscala**

Il motivo è evidente: per ogni istanza di una classe a specifica differenziata il Data Product deve contenere una indicazione precisa della scala di rilievo alla quale tale istanza è stata rilevata; infatti, in caso contrario non sarebbe possibile determinare la specifica alla quale tale istanza deve essere conforme. Ma, come mostrato al punto 8.3, la scala di un'istanza è definita solamente per le classi a istanze monoscala.
Dato che il vincolo tra le due suddivisioni delle classi è solamente quello citato, possono esistere 3 combinazioni possibili:

1. classi a specifica omogenea con istanze normali
2. classi a specifica differenziata con istanze monoscala
3. classi a specifica omogenea con istanze monoscala

Le prime due combinazioni corrispondono in maniera diretta alle considerazioni già fatte; la terza costituisce un'opzione applicabile per richiedere ad esempio di spezzare le istanze di certi oggetti sui limiti di zone di rilievo a scale diverse, anche se le proprietà della classe sono omogenee.

Conformità di un Data Product alla specifica di classi differenziate alle scale

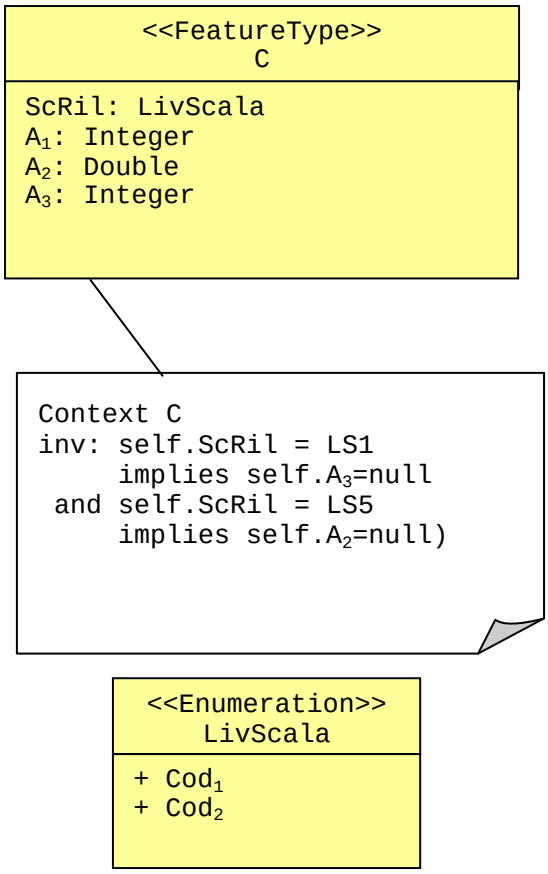
La regola di trasformazione di una classe GeoUML in un AS viene modificata dalla presenza di diversi popolamenti degli attributi alle diverse scale nel modo seguente:

Regola Attributi diversi ai diversi livelli di scala
Per ogni classe che possiede attributi popolati diversamente ai diversi livelli di scala viene generata una classe UML che contiene un attributo ScRil con dominio LivScala e tutti gli attributi popolati ad almeno un livello di scala; inoltre per ogni attributo Ax che non è popolato a livello di scala LSy viene definito un vincolo OCL con la seguente struttura:

```
Context C
inv: self.ScRil = LSy implies self.Ax=null
```

Esempio di applicazione della regola *Attributi diversi ai diversi livelli di scala*

classe C (abbreviazione - codice)
attributi:
attributi della classe
A_1: *Integer* *LS1 pop LS5 pop*
A_2: *Real* *LS1 pop LS5 no*
A_3: *Integer* *LS1 no LS5 pop*



tag values:
Cod_1: documentation = LS1
Cod_2: documentation = LS5

Regola Domini diversi ai diversi livelli di scala
Per ogni classe che possiede attributi con domini popolati diversamente ai diversi livelli di scala viene generata una classe UML che contiene un attributo ScRil con dominio LivScala e tutti gli attributi popolati ad almeno un livello di scala; inoltre per ogni attributo Ax il cui dominio è popolato al livello di scala LSy con i valori (Y_1, …, Y_n) e al livello di scala LSz con i valori (Z_1, …, Z_m) viene definito un vincolo OCL con la seguente struttura:

```
Context C
inv: self.ScRil = LSy implies self.Ax in Set{Y1,…,Yn}
     and
     (ScRil = LSz implies self.Ax in Set{Z1,…,Zm})
```

Effetto dei livelli di popolamento delle classi sui ruoli

Il popolamento delle classi ha il seguente impatto sull'interpretazione del vincolo di obbligatorietà (cardinalità minima) dei ruoli di un'associazione:

> data una classe *C* con un ruolo *r* verso *C'* con cardinalità minima 1: se la classe C' è non popolata almeno ad un livello di scala la cardinalità minima viene considerata opzionale.

Commento
Un'associazione semantica non dipende dalle geometrie delle classi, tuttavia il non popolamento di una classe ad almeno uno dei livelli di scala determina una riduzione dell'insieme degli oggetti di una classe, limitando le associazioni possibili.

Valutazione dei vincoli

In taluni casi il mancato popolamento di alcuni costrutti rende un vincolo non applicabile, quindi la conformità di un Data Product è ottenuta anche senza soddisfare quel vincolo.

Applicabilità del vincolo e popolamento delle classi

Classe vincolata.
Dato che ogni vincolo deve essere soddisfatto per ogni istanza della (selezione sulla) classe vincolata (e di tutte le sue sottoclassi), se la classe vincolata non è popolata il vincolo è automaticamente soddisfatto;

Classi vincolanti.
Il problema fondamentale nella valutazione dei vincoli si pone quando una classe vincolata è popolata ma non sono popolate alcune o tutte le classi vincolanti (includendo anche le loro sottoclassi) Si presentano i casi seguenti:

Vincoli di tipo perOgni (quantificazione universale)
Questi vincoli possono sempre essere applicati anche in assenza di classi vincolanti, perchè in assenza di istanze di classi vincolanti il vincolo è automaticamente soddisfatto.
L'esempio tipico di questo caso è costituito dai vincoli di disgiunzione DJ: se ogni istanza di una classe vincolata C deve essere disgiunta da ogni istanza di una classe vincolante C', allora se C' non è popolata il vincolo è sempre soddisfatto.

Vincoli che implicano una intersezione non nulla tra le parti interne delle classi vincolata e vincolante: Composizione e relazioni topologiche IN, CT, OV, EQ
La regola per queste situazioni è la seguente:

Regola interpretazione vincoli che implicano intersezione non nulla
Si consideri un vincolo del tipo
C vincolataDa V1, V2 ... Vn
Dove C è a istanze monoscala e vincolataDa sta per qualsiasi vincolo di composizione o basato sulle relazioni topologiche IN, CT, OV e EQ.
Un'istanza c della classe C deve soddisfare il vincolo se e solo se in almeno una delle classi V1, V2 ... Vn, la componente spaziale coinvolta nel vincolo è definita popolata al livello di scala di c, indicato dal valore dell'attributo ScRil di c.

Questa regola si basa sulla ipotesi, che costituisce una restrizione sulle modalità di rilievo alle scale definito nei capitolati, che se due istanze di oggetti **appartenenti a classi a istanze monoscala** si sovrappongono nella parte interna allora debbano essere ambedue rilevate alla stessa scala.
Ad esempio, le Unità Volumetriche che compongono un Edificio devono essere rilevate alla stessa scala dell'Edificio, i marciapiedi che compongono un'area stradale devono essere rilevati alla stessa scala dell'area stradale, ecc....

Altri vincoli
Per tutti gli altri vincoli il vincolo deve essere sempre soddisfatto indipendentemente dai livelli di scala.
Nella disgiunzione di vincoli topologici se un vincolo non è applicabile viene eliminato dalla disgiunzione.

Applicabilità del vincolo e popolamento degli altri costrutti

Per le classi popolate che partecipano a un vincolo è necessario applicare le seguenti regole:

1. Se una componente spaziale o un attributo a tratti o sottoaree che compare nel vincolo non è popolato a nessun livello di scala, il vincolo non è applicabile.
2. Se un attributo di classe o di attributo geometrico che compare nelle selezioni di un vincolo non risulta popolato a nessun livello di scala, il vincolo non viene considerato applicabile.
3. Gli attributi che compaiono nelle selezioni dei vincoli che sono popolati almeno ad un livello di scala si comportano nel modo seguente:
   - il loro valore deve essere considerato nullo per tutte le istanze rilevate ai livelli di scala ai quali l'attributo non è popolato;
   - la selezione viene valutata in base alle regole generali applicate per il valore nullo (vedi sezione 6.2.4).

Appendice A – Traduzione dei vincoli in OCL

A.1. Introduzione

In questa appendice si presentano le definizioni formali in OCL di tutti i vincoli di integrità spaziale presentati nel capitolo 6.
Si riporta per ogni template di vincolo una breve descrizione, la sintassi del template e la sua definizione in OCL. I template applicati ai vincoli usati in una specifica di contenuto produconoi vincoli in OCL da precisare nell'AS corrispondente alla specifica in conformità alle regole previste dallo standard ISO 19109 per la redazione di AS.

A.2. Vincolo topologico esistenziale

Vincolo esistenziale di base
Definizione dei simboli:
Date due classi X e Y contenenti almeno un attributo geometrico ciascuna, rispettivamente g e f, il vincolo topologico esistenziale da X verso Y, basato sulla disgiunzione di relazioni DJ_R = {rel_1,...,rel_n} si definisce come segue:

Sintassi:

vincolo X.g (rel_1 | … | rel_n) *esiste* Y.f

Template OCL:

```
ExistentialTopoConstraint (X, g, Y, f, DJ_R) ≡
context X
inv: Y.allInstances ->
     exists(a:Y | self.g.check(DJ_R, a.f))
```

Se uno o entrambi gli attributi geometrici `f` e g sono di tipo GU_C*SurfaceB3D è necessario specificare la componente considerata, ossia l'attributo `B3D` o l'attributo `superficie`; nella definizione sintattica si deve sostituire `X.g` (`Y.f`) con `X.g.B3D` (`Y.f.B3D`) o con `X.g.superficie` (`Y.f.superficie`) e contestualmente nel template OCL vanno sostituito `self.g` con `self.g.B3D` (`self.g.superficie`) e `a.f` con `a.f.B3D` (`a.f.superficie`).

È possibile far lavorare il vincolo anche sulla geometria di un attributo a tratti, a tratti sul contorno, a eventi o a sottoaree, in tal caso occorre sostituire nella specifica del vincolo l'attributo geometrico con la chiamata di una delle funzioni che restituisce i tratti, gli eventi o le sottoaree a seconda del tipo di attributo (ad esempio, le funzioni `TrattiDi_A()` e `SottoareeDi_A()`, dove `A` è il nome dell'attributo a tratti o a sottoaree).
Nel caso in cui l'attributo geometrico di una o entrambe le classi coinvolte sia di tipo GU_C*SurfaceB3D si dovrà specificare la funzione che ritorna i tratti sul contorno della superficie 2D o 3D.
Il vincolo si riformula come segue per gli attributi a tratti (nel seguito si fa riferimento all'attributo a tratti, tuttavia ogni vincolo può essere riformulato in modo simile per gli attributi a tratti sul contorno, a eventi e a sottoaree):

Variante su attributi a tratti (caso TR/TR)
Definizione dei simboli:
Date due classi X e Y contenenti almeno un attributo a tratti ciascuna, di nome rispettivamente a e b, il vincolo topologico esistenziale da X verso Y variante TR/TR, basato sulla disgiunzione di relazioni DJ_R={rel_1,...,rel_n} si definisce come segue:
Sintassi:

vincolo X.TrattiDi_a() (rel_1|…|rel_n) *esiste* Y.TrattiDi_b()

Template OCL:

ExistentialTopoConstraint$_{TR/TR}$ (X, a, Y, b, DJ_R) ≡

```
context X
inv: self.TrattiDi_a() -> forall(t:GU_Object|
          Y.allInstances.TrattiDi_b() ->
          exists(a:GU_Object | t.check(DJ_R, a)))
```

Si noti che quando il vincolo si riferisce alla geometria di un attributo a tratti si richiede che la disgiunzione di relazioni topologiche sia soddisfatta da tutti i tratti restituiti dalla funzione `TrattiDi_A()`. Quindi il risultato della valutazione del vincolo dipende anche dai valori restituiti da tale funzione.
Ovviamente il riferimento ai tratti può aversi anche solo sulla classe vincolata o solo sulla classe vincolante. Si mostrano di seguito per completezza anche queste due ulteriori varianti.

Variante su attributi a tratti (caso TR/GEO)
Definizione dei simboli:
Date due classi X e Y dove X contiene almeno un attributo a tratti (di nome a) e Y contiene un attributo geometrico f, il vincolo topologico esistenziale da X verso Y, variante TR/GEO, basato sulla disgiunzione di relazioni DJ_R={rel_1,...,rel_n} si definisce come segue:
Sintassi:

vincolo X.TrattiDi_a() (rel_1 | … | rel_n) *esiste* Y.f

Template OCL:

ExistentialTopoConstraint$_{TR/GEO}$ (X, a, Y, f, DJ_R) ≡

```
context X
inv: self.TrattiDi_a() -> forall(t:GU_Object|
          Y.allInstances.f ->
          exists(a:GU_Object | t.check(DJ_R, a)))
```

Variante su attributi a tratti (caso GEO/TR)
Definizione dei simboli:
Date due classi X e Y dove X contiene un attributo geometrico g e Y contiene un attributo a tratti (di nome b), il vincolo topologico esistenziale da X verso Y, variante GEO/TR, basato sulla disgiunzione di relazioni DJ_R={rel_1,...,rel_n} si definisce come segue:
Sintassi:

```
vincolo X.g (rel1 | … | reln) esiste Y.TrattiDi_b()
```

Template OCL:

```
ExistentialTopoConstraintGEO/TR (X, g, Y, b, DJ_R) ≡
context X
inv: Y.allInstances.TrattiDi_b() ->
        exists(a:GU_Object | self.g.check(DJ_R, a)))
```

Vincolo topologico esistenziale con selezioni
Una prima variante permette di selezionare gli oggetti delle classi coinvolte nel vincolo.

Vincolo esistenziale con selezioni
Definizione dei simboli:
Date due classi X e Y contenenti almeno un attributo geometrico ciascuna, rispettivamente g e f, il vincolo topologico esistenziale da X verso Y con selezioni, basato sulla disgiunzione di relazioni DJ_R={rel_1,...,rel_n} si definisce come segue:

Sintassi:

```
vincolo (σ1(X))X.g (rel1 | … | reln) esiste (σ2(X,Y))Y.f
```

Template OCL:

```
ExistentialTopoConstraintSEL (X,σ1(X),g,Y,σ2(X,Y),f,DJ_R) ≡

context X
inv: σ1(self) implies
     (Y.allInstances -> exists(a:Y| σ2(self, a) and
                         self.g.check(DJ_R, a.f)))
```

La clausola di selezione $\sigma_1(X)$ è una formula proposizionale del tipo [not](α_1 opLogico … opLogico α_n) con opLogico$\in${AND,OR} e α_i è una formula atomica del tipo: (X.a op X.b), (X.a op cost), (X.a = nullo) o (X.a = non nullo) dove entrambi gli attributi appartengono alla classe X e possono essere multivalore, op$\in${=,<>, <, >, ≥, ≤≠} e cost è un valore diverso da nullo. Nelle prime due formule in presenza di almeno un operando (X.a o X.b) multivalore la valutazione della formula è effettuata nel seguente modo: si generano tutte le combinazioni possibili (prodotto cartesiano) tra i valori degli operandi e la formula atomica sarà valutata "true" se esiste almeno una coppia di valori che soddisfa la condizione di confronto; si noti che nel caso di attributi monovalore ciò corrisponde a generare una sola coppia di valori da confrontare. Le ultime due formule verificano l'esistenza o meno del valore nullo nell'attributo.
$\sigma_2(X,Y)$ è una formula proposizionale analoga a $\sigma_1(X)$ dove α_i ammette anche una formula del tipo (Y.a op X.b) che coinvolge un attributo della classe vincolata.
Gli attributi coinvolti nelle formule sono quelli delle classi ad eccezione di quelli geometrici, a tratti (eventi, sottoaree); negli attributi con dominio enumerato gerarchico

non vengono considerati gli attributi aggiuntivi e nel caso di DataType è necessario specificare nella formula atomica lo specifico attributo.
Le formule di selezione presentate ammettono due casi:

1. **selezioni normali**: espressioni logiche di predicati semplici del tipo “attributo comparatore valore” ($\sigma_1(X)$, $\sigma_2(X,Y)$), dove attributo deve appartenere alla classe alla quale si applica la selezione;
2. **selezioni di join**: in questa forma si ammettono predicati del tipo “attributo comparatore classeVincolata.attributo” ($\sigma_2(X,Y)$), che permettono di legare gli oggetti della classe vincolante, utilizzabili per soddisfare il vincolo, all’oggetto considerato della classe vincolata.

Si richiama che il riferimento ad un attributo nel vincolo (ad esempio, X.a) deve utilizzare il codice alfanumerico della classe e può essere omesso se è chiaro la classe corrente della selezione (ad esempio, la classe X nella clausola $\sigma_1(X)$). Nel caso della selezione sulla classe vincolante può essere omesso il riferimento alla classe vincolante, ma non a quella vincolata; in particolare, il riferimento alla classe vincolata deve essere preceduto dal simbolo “$” nel caso in cui un vincolo coinvolga una classe sia come classe vincolata che come classe vincolante.

Oltre alla clausola di selezione applicata per restringere gli oggetti coinvolti nel vincolo è possibile esprimere anche la condizione di selezione per restringere i tratti coinvolti eventualmente in un vincolo (omettiamo per semplicità le varianti inerenti gli attributi a tratti sul contorno, a eventi e a sottoaree) la condizione delle funzioni `TrattiDi_A(cond_selezione)`. Tali funzioni quando compaiono in un vincolo consentono di precisare come parametro una clausola di selezione arricchita rispetto a quanto precisato nella loro definizione che viene poi fatta corrispondere a quella attesa.

La clausola di selezione $\sigma_{1a}(X,a)$ è una formula proposizionale del tipo [not](α_1 opLogico … opLogico α_n) con opLogico$\in${AND,OR} e α_i è una formula atomica che coinvolge sempre l’attributo a tratti X.a del tipo (X.a op cost), (X.a op X.b), (X.a = nullo) o (X.a = non nullo), dove op$\in${=, <>, <, >, ≥, ≤}, cost è un valore diverso da nullo; nel caso in cui l’attributo X.b sia multivalore la condizione (X.a op X.b) è trasformata in una disgiunzione di formule atomiche (X.a op valore), dove valore corrisponde in ogni formula ad uno dei valori dell’attributo multivalore X.b.
$\sigma_{2b}(X,Y,b)$ è una formula proposizionale analoga a $\sigma_{1a}(X,a)$ dove α_i ammette anche la formula (Y.b op X.c) dove Y.b è l’attributo a tratti e X.c un attributo della classe vincolata.
In entrambe le clausole di selezione si utilizzano attributi delle classi coinvolte, ad eccezione degli attributi geometrici o degli ulteriori attributi a tratti presenti. Inoltre i domini degli attributi possono essere di base, enumerati, enumerati gerarchici ignorando gli ulteriori attributi e nel caso di dominio DataType si deve esplicitare l’attributo considerato.
Anche in questo caso si applicano le regole sintattiche sull’uso del codice alfanumerico delle classi e sull’omissione di tale codice descritte per la selezione degli oggetti delle classi.
Di seguito si mostrano le tre versioni del vincolo con selezioni nei casi: tratti su tratti (TR/TR), attributo geometrico su tratti (GEO/TR) e trat su attributo geometrico (TR/GEO).

Vincolo esistenziale su tratti con selezioni (TR/TR, GEO/TR, TR/GEO)
Definizione dei simboli:
Date due classi X e Y contenenti un attributo geometrico ciascuna, rispettivamente g e f, e un attributo a tratti ciascuna, rispettivamente a e b, si definisce vincolo topologico esistenziale da X verso Y con selezioni, variante TR/TR, GEO/TR, TR/GEO, basato sulla disgiunzione di relazioni DJ_R = {rel_1,...,rel_n) il seguente vincolo:

TR/TR
Sintassi:

vincolo $(\sigma_1(X))X.TrattiDi\_a(\sigma_{1a}(X,a))$
$(rel_1, \ldots, rel_n)$ *esiste*
$(\sigma_2(X,Y))Y.TrattiDi\_b(\sigma_{2b}(X,Y,b))$

e
Template OCL:

$ExistentialTopoConstraint_{TR/TR}^{SEL}$
$(X, \sigma_1(X), a, \sigma_{1a}(X,a), Y, \sigma_2(X,Y), b, \sigma_{2b}(X,Y,b), DJ\_R) \equiv$

```
context X
inv: σ1(self) implies
  self.TrattiDi_a("σ1a(X,a)")->
    forall(t:GU_Object|
       Y.allInstances->
       select(y:Y| σ2(self,y)).TrattiDi_b("σ2b(X,Y,b)")->
       exists(a:GU_Object| t.check(DJ_R, a))
    )
```

GEO/TR
Sintassi:

vincolo $(\sigma_1(X))X.g$
$(rel_1, \ldots, rel_n)$ *esiste*
$(\sigma_2(X,Y))Y.TrattiDi\_b(\sigma_{2b}(X,Y,b))$

Template OCL:

$ExistentialTopoConstraint_{GEO/TR}^{SEL}$
$(X, \sigma_1(X), g, Y, \sigma_2(X,Y), b, \sigma_{2b}(X,b), DJ\_R) \equiv$

```
context X
inv: σ1(self) implies
     Y.allInstances->
     select(y:Y| σ2(self,y)).TrattiDi_b("σ2(X,Y,b)")->
     exists(a:GU_Object | self.g.check(DJ_R, a))
```

TR/GEO

Sintassi:

vincolo (σ_1(X))X.TrattiDi_a(σ_{1a}(X,a))
(rel_1, …, rel_n) *esiste*
(σ_2(X,Y))Y.f

Template OCL:

$ExistentialTopoConstraint_{TR/GEO}^{SEL}$
(X, σ_1(X), a, σ_{1a}(X,a), Y, σ_2(X,Y), f, DJ_R) ≡

```
context X
inv: σ1(self) implies
  self.TrattiDi_a("σ1(X,a)")->
    forall(t:GU_Object|
          Y.allInstances->exists(a:Y| σ1(self,a)) and
                                    t.check(DJ_R, a))
    )
```

Vincolo topologico esistenziale sulla frontiera o sulla proiezione planare

Vincolo esistenziale su frontiera o proiezione planare

Definizione dei simboli:

Date due classi X e Y contenenti almeno un attributo geometrico ciascuna, rispettivamente g e f, il vincolo esistenziale sulla frontiera da X verso Y, basato sulla disgiunzione di relazioni DJ_R = {rel_1,...,rel_n} si definisce come segue:

Sintassi:

vincolo X.g.*BND* (rel_1 | … | rel_n) *esiste* Y.f

Template OCL:

$ExistentialTopoConstraint^{B/-}$ (X, g, Y, f, DJ_R) ≡

```
context X
inv: Y.allInstances ->
exists(a:Y | self.g.boundary().check(DJ_R, a.f)
```

si definisce inoltre il vincolo esistenziale sulla proiezione da X verso Y, basato sulla disgiunzione di relazioni DJ_R = {rel_1,...,rel_n} come segue:

Sintassi:

vincolo X.g.*PLN* (rel_1 | … | rel_n) *esiste* Y.f

Template OCL:

$ExistentialTopoConstraint^{P/-}$
(X, g: GU_Object, Y, f: GU_Object, DJ_R) ≡

```
context X
inv: Y.allInstances ->
exists(a:Y | self.g.planar().check(DJ_R, a.f))
```

Similmente si definiscono le varianti con la funzione `boundary()` o `planar()` applicate sull'attributo `f` della classe vincolante:

$ExistentialTopoConstraint^{-/B}$,

`ExistentialTopoConstraint`$^{-/P}$
o su entrambi gli attributi geometrici
`ExistentialTopoConstraint`$^{B/B}$,
`ExistentialTopoConstraint`$^{P/P}$
così come tutte le varianti che si ottengono dall'applicazione in cascata delle due funzioni.

Si ricorda che la funzione .PLN applicata ad una geometria definita nello spazio 2D non modifica la geometria e che, applicata ad un oggetto definito nello spazio 3D, lo trasforma in un oggetto dello spazio 2D.
E' possibile combinare questa variante a quella che ammette la selezione, gli attributi a tratti o una componente di una superficie con frontiera in 3D.

Vincolo topologico collegato ad una associazione
Con questa variante si considerano, ai fini del soddisfacimento del vincolo, solo gli oggetti della classe vincolante che sono legati all'oggetto da verificare attraverso un'associazione specificata nello schema.

Vincolo esistenziale collegato ad una associazione
Definizione dei simboli:
Siano date due classi X e Y contenenti almeno un attributo geometrico ciascuna, di nome rispettivamente g e f, e tra le quali esista un'associazione, dove il ruolo di Y sia r. Il vincolo topologico esistenziale da X verso Y, basato sulla disgiunzione di relazioni $DJ\_R = \{rel_1,...,rel_n\}$ *e agganciato all'associazione attraverso il ruolo r si definisce come segue:*

Sintassi:
vincolo `X.g (rel`$_1$ `| … |` `rel`$_n$`)` *esiste* `X.r.f`

Template OCL:
`ExistentialTopoConstraint`^{A}
`(X, g, r, Y, f, DJ_R)` ≡

```
context X
inv: self.r -> exists(a:Y| self.g.check(DJ_R, a.f))
```

A.3. Vincolo topologico unione

Vincolo unione

Definizione dei simboli:

Date due classi X e Y contenenti almeno un attributo geometrico ciascuna, rispettivamente g e f, il vincolo topologico con unione da X verso Y, basato sulla disgiunzione di relazioni DJ_R={rel$_1$,...,rel$_n$} si definisce come segue:

Sintassi:

vincolo X.g (rel$_1$ | … | rel$_n$)*unione* Y.f

Template OCL:

```
UnionTopoConstraint
     (X, g, Y, f, DJ_R) ≡

context X
inv: self.g.check(DJ_R, Y.allInstances ->
          iterate(a:Y, acc: GU_Object = ∅ |
                       acc.gUnion(a.f)))
```

Anche per il vincolo topologico su unione esistono le varianti presentate per il vincolo topologico esistenziale, vale a dire: la versione con selezione, la versione che si riferisce al boundary() e planar(), quella che si riferisce ad un'associazione.

Vincolo unione con selezioni

Definizione dei simboli:

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 con selezione, basato sulla disgiunzione di relazioni DJ_R={rel$_1$,...,rel$_n$} il seguente vincolo:

Sintassi:

vincolo (σ_1(X))X.g (rel$_1$ | … | rel$_n$) *unione* (σ_2(X,Y))Y.f

Template OCL:

```
UnionTopoConstraint^SEL(X, σ1(X), g, Y, σ2(X,Y), f, DJ_R) ≡

context X
inv: σ1(self) implies
  self.g.check(DJ_R,
    Y.allInstances->select(a:Y| σ2(self,a))->
    iterate(b:Y, acc: GU_Object = ∅ |
                 acc.gUnion(b.f)))
```

E' possibile applicare il vincolo unione anche su attributi a tratti, considerando l'attributo a tratti solo per la classe vincolata, visto che sulla vincolante si richiede comunque l'unione delle geometrie. Vengono direttamente specificate le varianti con selezione, quelle senza selezione si ottengono semplicemente togliendo le condizioni di selezione.

Vincolo unione su attributi a tratti con selezione (TR/TR, GEO/TR, TR/GEO)
Definizione dei simboli:
Date due classi X e Y contenenti un attributo geometrico ciascuna, rispettivamente g e f, e un attributo a tratti ciascuna, rispettivamente a e b, si dice vincolo topologico con unione da X verso Y con selezione, variante TR/TR, GEO/TR, TR/GEO, basato sulla disgiunzione di relazioni DJ_R={rel_1,...,rel_n} il seguente vincolo:

TR/GEO
Sintassi:

vincolo (σ_1(X))X.TrattiDi_a(σ_1(X,a))
(rel_1 | … | rel_n) *unione* (σ_2(X,Y))Y.f

Semantica:

UnionTopoConstraint$^{SEL}_{TR/GEO}$
(X, σ_1(X), a, σ_1(X,a), Y, σ_2(X,Y), f, DJ_R) ≡
context X
inv: σ_1(self) implies
self.TrattiDi_a("σ_1(X,a)")-> forall(t:GU_Object|
t.check(DJ_R,
Y.allInstances-> select(a:Y|σ_2(self,a)).f->
iterate(b:GU_Object,
acc: GU_Object = ∅ |
acc.gUnion(b))))

GEO/TR
Sintassi:

vincolo (σ_1(X))X.g (rel_1 | … | rel_n)
unione (σ_2(X,Y))Y.TrattiDi_b(σ_{2b}(X,Y,b))

Semantica:

UnionTopoConstraint$^{SEL}_{GEO/TR}$
(X, σ_1(X), g, Y, σ_2(X,Y), b, σ_{2b}(X,Y,b), DJ_R) ≡
context X
inv: σ_1(self) implies
self.g.check(DJ_R, Y.allInstances->
select(a:Y| σ_2(self,a)).TrattiDi_b("σ_2(X,Y,b)")->
iterate(b:GU_Object,
acc: GU_Object = ∅ |
acc.gUnion(b))))

TR/TR

Sintassi:

vincolo (σ_1(X))X.TrattiDi_a(σ_1(X,a)) (rel_1 | … | rel_n)
unione (σ_2(X,Y))Y.TrattiDi_b(σ_{2b}(X,Y,b))

Semantica:

UnionTopoConstraint$^{SEL}_{TR/TR}$
(X, σ_1(X), a, σ_1(X,a), Y, σ_2(X,Y), b, σ_{2b}(X,Y,b), DJ_R)≡

```
context X
inv: σ1(self) implies
  self.TrattiDi_a("σ1(X,a)")-> forall(t:GU_Object|
       t.check(DJ_R, Y.allInstances-> select(a:Y|
            σ2(self,a)).TrattiDi_b("σ2b(X,Y,b)")->
                iterate(b:GU_Object,
                        acc: GU_Object = ∅ |
                        acc.gUnion(b))))
```

A.4. Vincolo topologico universale

Vincolo universale

Definizione dei simboli:

Date due classi X e Y contenenti almeno un attributo geometrico ciascuna, rispettivamente g e f, il vincolo topologico universale da X verso Y, basato sulla disgiunzione di relazioni DJ_R={rel_1,...,rel_n} si definisce come segue:

Sintassi:

vincolo X.g (rel_1 | … | rel_n) *perOgni* Y.f

Template OCL:

```
UniversalTopoConstraint
     (X, g, Y, f, DJ_R) ≡

context X
inv: Y.allInstances->forAll(a:Y| self.g.check(DJ_R, a.f))
```

Vincolo universale con selezioni

Definizione dei simboli:

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 con selezione, basato sulla disgiunzione di relazioni DJ_R={rel_1,...,rel_n} il seguente vincolo:

Sintassi:

vincolo (σ_1(X))X.g (rel_1 | … | rel_n) *perOgni* (σ_2(X,Y))Y.f

Template OCL:

```
UniversalTopoConstraint^SEL
     (X, σ1(X), g, Y, σ2(X,Y), f, DJ_R) ≡

context X
inv: σ1(self) implies
     Y.allInstances -> select(a:Y| σ2(self,a)->
     forAll(b:Y| self.g.check(DJ_R, b.f))
```

Inoltre è possibile applicare il vincolo universale anche su attributi a tratti, a eventi o a sottoaree con la seguente semantica.

Vincolo universale su attributi a tratti con selezioni (TR/TR, GEO/TR, TR/GEO)
Definizione dei simboli:
Date due classi X e Y contenenti almeno un attributo a tratti ciascuna, rispettivamente a e b, si dice vincolo topologico universale da X verso Y con selezioni, variante TR/TR, GEO/TR, TR/GEO, basato sulla disgiunzione di relazioni DJ_R={rel_1,...,rel_n} il seguente vincolo:

TR/TR

Sintassi:

vincolo (σ_1(X))X.aTratti_a(σ_{1a}(X,a))
(rel_1 | … | rel_n) *perOgni*
(σ_2(X,Y))Y.aTratti_b(σ_{2b}(X,Y,b))

Template OCL:

UniversalTopoConstraint$^{SEL}_{TR/TR}$
(X, σ_1(X), a, σ_{1a}(X,a), Y, σ_2(X,Y), b, σ_{2b}(X,Y,b), DJ_R) ≡
context X
inv: σ_1(self) implies
self.TrattiDi_a("σ_{1a}(X,a)")->
forall(t:GU_Object |
Y.allInstances->
select(y:Y| σ_2(self,Y)).TrattiDi_b("σ_{2b}(X,Y,b)")->
forAll(c:GU_object| t.check(DJ_R, c))
)

GEO/TR

Sintassi:

vincolo (σ_1(X))X.g
(rel_1 | … | rel_n) *perOgni*
(σ_2(X,Y))Y.aTratti_b(σ_{2b}(X,Y,b))

Template OCL:

UniversalTopoConstraint$^{SEL}_{GEO/TR}$
(X, σ_1(X), g, Y, σ_2(X,Y), b, σ_{2b}(X,Y,b), DJ_R) ≡
context X
inv: σ_1(self) implies
Y.allInstances->
select(y:Y| σ_2(self,Y)).TrattiDi_b("σ_{2b}(X,Y,b)")->
forAll(c:GU_object | self.g.check(DJ_R, c))

TR/GEO

Sintassi:

vincolo (σ_1(X))X.aTratti_a(σ_{1a}(X,a))
(rel$_1$ | … | rel$_n$) *perOgni*
(σ_2(X,Y))Y.f

Template OCL:

```
UniversalTopoConstraint^SEL_TR/GEO
     (X, σ1(X), a, σ1a(a,X), Y, σ2(X,Y), f, DJ_R) ≡
context X
inv: σ1(self) implies
self.TrattiDi_a("σ1a(X,a)") ->
  forall(t:GU_Object |
    Y.allInstances->select(b:Y| σ2(self,b))->
      forAll(c:Y | t.check(DJ_R, c.f))
  )
```

A.5. Vincolo di composizione

Vincolo di composizione
Definizione dei simboli:
Data una classe Y con attributo geometrico f ed una classe X con attributo geometrico g, il vincolo compostoDa da Y verso X si definisce nel seguente modo:
Sintassi:

```
vincolo Y.f compostoDa X.g
```

Template OCL:

```
ComposedOfConstraint(Y, f, X, g) ≡

context Y
inv: self.f.Equals(X.allInstances.g ->
     select(a:GU_Object| self.f.Contains(a) or
                         self.f.Equals(a) ) ->
     iterate(b:GU_Object, acc: GU_Object = ∅ |
             acc.gUnion(b))
```

Se uno o entrambi gli attributi geometrici `f` e g sono di tipo GU_C*SurfaceB3D è necessario specificare quale sia la componente considerata, ossia l'attributo B3D (superficie); pertanto nella definizione sintattica del vincolo nel caso di entrambi gli attributi si deve sostituire `X.g` con `X.g.B3D` (`superficie`) e `Y.f` con `Y.f.B3D` (`superficie`) e contestualmente nel template OCL va sostituito `self.f` con `self.f.B3D` (`superficie`), `X.AllInstances.g` con `X.AllInstances.g.B3D` (`superficie`) e `self.f.Contains(a)` con `self.f.B3D`(`superficie`).`Contains(a)`.

Si mostra di seguito la definizione del vincolo compostoDa con selezione gli altri vincoli di modificano in presenza di selezione nello stesso modo.

Vincolo di composizione con selezioni
Definizione dei simboli:
Data una classe Y con attributo geometrico f ed una classe X con attributo geometrico g, il vincolo compostoDa (copertoDa) con selezione si definisce nel seguente modo:
Sintassi:

```
vincolo (σ1(Y))Y.f compostoDa (σ2(Y,X))X.g
```

Template OCL:

```
ComposedOfConstraint^SEL(Y, σ1(Y), f, X, σ2(Y,X), g)
context Y
inv: σ1(self) implies (self.f.Equals(
     X.allInstances->select(x:X| σ2(self,x)).g->
     select(a:GU_Object| self.f.Contains(a)
                         or self.f.Equals(a))->
     iterate(b:GU_Object, acc: GU_Object = ∅ |
             acc.gUnion(b)))
```

Il vincolo si riformula come segue per gli attributi a tratti:

Vincolo di composizione su attributi a tratti con selezioni (casi TR/TR, GEO/TR e

TR/GEO)

Definizione dei simboli:

Date due classi Y e X contenenti un attributo geometrico ciascuna, rispettivamente f e g, e un attributo a tratti ciascuna, rispettivamente b e a, il vincolo compostoDa(copertoDa) da Y verso X , variante TR/TR, GEO/TR, TR/GEO, si definisce nel seguente modo:

TR/TR

Sintassi:

vincolo (σ_1(Y))Y.TrattiDi_b(σ_{1b}(Y,b)) *compostoDa*
(σ_2(Y,X))X.TrattiDi_a(σ_{2a}(Y,X,a))

Template OCL:

ComposedOfConstraint$^{SEL}_{TR/TR}$(Y, σ_1(Y), b, σ_{1b}(Y,b),
X, σ_2(Y,X), a, σ_{2a}(Y,X,a))

```
context Y
inv: σ1(self) implies
self.TrattiDi_b("σ1b(Y,b)")->
 forall(t: GU_Object| t.Equals(
  X.allInstances->
  select(x:X| σ2(self,x)).TrattiDi_a("σ2a(Y,X,a)")->
  select(c:GU_Object| t.Contains(c)or t.Equals(c))->
        iterate(d:GU_Object, acc: GU_Object = ∅ |
                        acc.gUnion(d))
        )
```

GEO/TR

Sintassi:

vincolo (σ_1(Y))Y.f *compostoDa*
(σ_2(Y,X))X.TrattiDi_a(σ_{2a}(Y,X,a))

Template OCL:

ComposedOfConstraint$^{SEL}_{GEO/TR}$(Y, σ_1(Y), f, X, σ_2(Y,X),
a, σ_{2a}(Y,X,a))

```
context Y
inv: σ1(self) implies
self.f.Equals(
 X.allInstances->
   select(x:X| σ2(self,x)).TrattiDi_a("σ2a(Y,X,a)")->
      select(c:GU_Object| self.f.Contains(c)
                          or self.f.Equals(c))->
        iterate(d:GU_Object, acc: GU_Object = ∅ |
                             acc.gUnion(d))
             )
```

TR/GEO
Sintassi:

```
vincolo (σ1(Y))Y.TrattiDi_b(σ1b(Y,b)) compostoDa
        (σ2(Y,X))X.g
```

Template OCL:

```
ComposedOfConstraint^SEL_TR/GEO(Y, σ1(Y), b, σ1b(Y,b),
     X, σ2(Y,X), g)
context Y
inv: σ1(self) implies
self.TrattiDi_b("σ1b(Y,b)")->
  forall(t: GU_Object| t.Equals(
    X.allInstances ->
      select(c:X| σ2(self,c)
          and (t.Contains(c.g)or t.Equals(c.g)).g)->
       iterate(d:GU_Object, acc: GU_Object = ∅ |
                            acc.gUnion(d)))
  )
```

Di seguito è data la definizione formale dei vincoli di composizione sulla frontiera e sulla proiezione.

Vincolo di composizione su frontiera e proiezione planare
Definizione dei simboli:
Data una classe Y con attributo geometrico f ed una classe X con attributo geometrico g, il vincolo di composizione sulla frontiera (o sulla proiezione) si definisce nel seguente modo:

Sintassi:

```
vincolo Y.f.BND compostoDa X.g
```

Template OCL:

```
ComposedOfConstraint^B-(Y, f, X, g)

context Y
inv: self.f.boundary().Equals(
 X.allInstances.g->
  select(a:GU_Object| self.f.boundary().Contains(a)
                   or self.f.boundary().Equals(a))->
   iterate(b:GU_Object, acc: GU_Object = ∅ |
                        acc.gUnion(b)))
```

Similmente si definiscono le varianti con la funzione `boundary()` o `planar()` applicate sull'attributo `f` e g rispettivamente:

`ComposedOfConstraint`$^{P/-}$, `ComposedOfConstraint`$^{-/B}$

o su entrambi gli attributi geometrici

`ComposedOfConstraint`$^{B/B}$, `ComposedOfConstraint`$^{P/P}$

così come tutte le varianti che si ottengono dall'applicazione in cascata delle due funzioni.

Infine è data la definizione formale del vincolo di composizione basato sulle associazioni, nel quale i vincoli di composizione non devono fare riferimento a 2 classi indipendenti e alla

proprietà di contenimento geometrico, ma devono appoggiarsi ad un'associazione che le collega. Ciò significa che gli oggetti della classe vincolante che devono partecipare al vincolo sono quelli che partecipano all'associazione. Il vincolo di composizione viene definito come segue.

Vincolo di composizione su associazione
Definizione dei simboli:
Data una classe Y con attributo geometrico f ed un ruolo r verso una classe X con attributo geometrico g, il vincolo compostoDa su associazione da Y verso X è definito come segue:
Sintassi:

```
vincolo Y.f compostoDa Y.r.g
```

Template OCL:

```
ComposedOfOnAssociation(Y, f, r, g) ≡

context Y
inv: self.f.Equals(self.r.g ->
       iterate(b:GU_Object,acc: GU_Object = ∅ |
          acc.gUnion(b)))
```

Come si può notare l'unica differenza rispetto al vincolo non collegato ad una associazione è il fatto che, invece di partire da tutti gli oggetti di `X` (`X.allInstances`), si parte solo dalle istanze raggiungibili attraverso il ruolo r dell'associazione (`self.r`).

A.6. Vincolo di appartenenza

Vincolo di appartenenza dj-IN

Definizione dei simboli:

Data una classe X con attributo geometrico g ed una classe Y con attributo geometrico f, il vincolo dj-IN da X verso Y è definito come segue:

Sintassi:

```
vincolo X.g dj-IN Y.f
```

Template OCL:

```
dj-IN(X, g, Y, f) ≡

context X
inv: Y.allInstances.f->
      exists(a:GU_Object|self.g.In(a)
                         or self.g.Equals(a))
     and
     X.allInstances ->
     select(x | (x.OID ≠ self.OID)).g->
     forall(v: GU_Object |brotherIN(v, self.g, Y, f)
        implies
         (v.Disjoint(self.g) or
          v.Touch(self.g) and v.Touch(self.g.boundary())))


brotherIN(x1, x2, Y, f)
  ≡ Y.allInstances ->
     exists(y:Y| ((x1.In(y.f)) or (x1.Equals(y.f))
                 and
                 ((x2.In(y.f) or (x2.Equals(y.f))
```

Vincolo di appartenenza qdj-IN
Definizione dei simboli:
Data una classe X con attributo geometrico g ed una classe Y con attributo geometrico f, il vincolo qdj-IN da X verso Y è definito come segue:
Sintassi:

```
vincolo X.g qdj-IN Y.f
```

Template OCL:

```
qdj-IN (X, g, Y, f) ≡

context X
inv:Y.allInstances.f->
   exists(a: GU_Object| self.g.In(a) or
                        self.g.Equals(a))
    and
    X.allInstances ->
    select(x | (x.OID ≠ self.OID).g->
    forall(v: GU_Object | brotherIN(v, self.g, Y, f)
       implies
            (v.Disjoint(self.g) or
             v.Touch(self.g) or
             v.Cross(self.g)))
```

dove brotherIN() è la funzione definita nel caso precedente.

Anche per i vincoli di appartenenza sono applicabili alcune delle varianti già introdotte per i vincoli topologici.

Vincolo di appartenenza con selezioni
Definizione dei simboli:
Data una classe X con attributo geometrico g ed una classe Y con attributo geometrico f, il vincolo dj-IN con selezioni è definito come segue:
Sintassi:

```
vincolo (σ1(X))X.g dj-IN (σ2(X,Y))Y.f
```

Template OCL:

```
dj-IN(X, σ1(X), g, Y, σ2(X,Y), f) ≡

context X
inv: σ1(self) implies (
(Y.allInstances-> select(y:Y| σ2(self,y)).f->
  exists(a:GU_Object| self.g.In(a) or
                      self.g.Equals(a)))
and
(X.allInstances->
  select(x:X| σ1(x) and (x.OID ≠ self.OID)).g ->
   forall(v: GU_Object | brotherIN(v, self.g, Y, f)
    implies
      (v.Disjoint(self.g) or
       v.Touch(self.g) and v.Touch(self.g.boundary()))))
```

dove brotherIN è la funzione definita in sezione 6.2.3

Tale variante si propaga banalmente al vincolo *qdj-IN*.

Quando si vuole che il vincolo sia riferito alla geometria di un attributo a tratti occorre

sostituire nella specifica del vincolo l'attributo geometrico con la chiamata di una della funzione: `TrattiDi_A()`, dove A è il nome dell'attributo a tratti.
Il vincolo si riformula come segue per gli attributi a tratti:

Vincolo di appartenenza su attributi a tratti con selezioni (casi TR/TR, GEO/TR e TR/GEO)

Definizione dei simboli:

Date due classi X e Y contenenti un attributo geometrico ciascuna, rispettivamente g e f, e un attributo a tratti ciascuna, rispettivamente a e b, il vincolo dj-IN da X verso Y, variante TR/TR, GEO/TR, TR/GEO con selezioni, si definisce nel seguente modo:

TR/TR

Sintassi:

vincolo $\sigma_1(X)X.TrattiDi\_a(\sigma_{1a}(X,a))$ *dj-IN* $\sigma_2(X,Y)Y.TrattiDi\_b(\sigma_{2b}(X,Y,b))$

Template OCL:

$dj\text{-}IN_{TR/TR}^{SEL}(X, \sigma_1(X), a, \sigma_{1a}(X,a), Y, \sigma_2(X,Y), b, \sigma_{2b}(X,Y,b)) \equiv$

```
context X
inv: σ1(self) implies
(self.TrattiDi_a("σ1a(X,a)")->
  forall(t: GU_Object|
   Y.allInstances->
   select(y:Y| σ2(self,y)).TrattiDi_b("σ2b(X,Y,b)")->
      exists(a:GU_Object| t.In(a) or t.Equals(a))
   and
   X.allInstances->
    select(x:X| σ1(x)).TrattiDi_a("σ1a(X,a)")->
      select (v: GU_Object | not (v.sameObj(t))
           or (v.sameObj(t)) and
               v.sameValueTratto(t))->
       forall(c: GU_Object|
          brotherIN(c, t, Y, f)
           implies
          (c.Disjoint(t) or
           c.Touch(t)and c.Touch(t.boundary())))))
```

Dove le funzioni `brotherIN()`è quella definita in sezione 6.2.3, `sameObj()` verifica se i tratti confrontati appartengono allo stesso oggetto e infine `sameValueTratto()` verifica se i due tratti hanno lo stesso valore di attributo a tratti associato.

GEO/TR

Sintassi:

vincolo (σ_1(X))X.g *dj-IN*
(σ_2(X,Y))Y.TrattiDi_b(σ_{2b}(X,Y,b))

Template OCL:

dj-IN$_{GEO/TR}^{SEL}$(X, σ_1(X), g, Y, σ_2(X,Y), b, σ_{2b}(X,Y,b)) ≡

```
context X
inv: σ1(self) implies (
Y.allInstances->
  select(y:Y| σ2(self,y)).TrattiDi_b("σ2b(X,Y,b)")->
    exists(a:GU_Object| self.In(a) or
                        self.Equals(a))
and
X.allInstances->
 select(x:X| σ1(x) and x.OID ≠ self.OID)).g->
  forall(c: GU_Object| brotherIN(c, self.g, Y, f)
   implies
    (c.Disjoint(self.g) or
     c.Touch(self.g) and c.Touch(self.g.boundary())))
```

TR/GEO

Sintassi:

vincolo (σ_1(X))X.TrattiDi_a(σ_{1a}(X,a)) *dj-IN*
(σ_2(X,Y))Y.f

Template OCL:

dj-IN$_{TR/GEO}^{SEL}$(X, σ_1(X), a, σ_{1a}(X,a), Y, σ_2(X,Y), f) ≡

```
context X
inv: σ1(self) implies
(self.TrattiDi_a("σ1a(X,a)")->
  forall(t: GU_Object |
    Y.allInstances->select(y:Y| σ2(self,y)).f->
      exists(a:GU_Object| t.In(a) or t.Equals(a))
    and
    X.allInstances->
     select(x:X| σ1(x)).TrattiDi_a("σ1a(X,a)")->
       select (v: GU_Object | not (v.sameObj(t))
            or (v.sameObj(t) and
                v.sameValueTratto(t))->
         forall(c: GU_Object|
          brotherIN(c, t, Y, f)
           implies
           (c.Disjoint(t) or
            c.Touch(t) and c.Touch(t.boundary()))))
```

Dove la funzione `brotherIN()` è quella definita in sezione 6.2.3 e le funzioni `sameObj()` e `sameValueTratto()` sono quelle definite precedentemente in questa sezione.

Vincolo di appartenenza su frontiera e proiezione planare
Definizione dei simboli:
Data una classe X con attributo geometrico g ed una classe Y con attributo geometrico f, il vincolo dj-IN sulla frontiera o proiezione planare, è definito come segue:
Sintassi:

vincolo X.g.*BND* *dj-IN* Y.f

Template OCL:

dj-IN^{B-}(X, g, Y, f) ≡

```
context X
inv: Y.allInstances.f->
  exists(a:GU_Object| self.g.boundary().In(a)
                   or self.g.boundary().Equals(a)))
  and X.allInstances->
       select (x | (x.OID ≠ self.OID)).g.boundary->
        forall(c: GU_Object|
            brotherIN(c, self.g.boundary(), Y, f)
            implies
            (c.Disjoint(self.g.boundary()) or
             c.Touch(self.g.boundary())))
```

Sintassi:

vincolo X.g *dj-IN* Y.f.*PLN*

Semantica:

dj-IN^{-P}(X, g, Y, f) ≡

```
context X
inv: Y.allInstances.f.planar()->
  exists(a:GU_Object| self.g.In(a) or
                      self.g.Equals(a)))
  and
  X.allInstances.g->
   select(x:X| x.OID ≠ self.OID).g ->
    forall(v: GU_Object | brotherIN(v, self.g, Y, f)
    implies
      (v.Disjoint(self.g) or
       v.Touch(self.g)and v.Touch(self.g.boundary())))
```

dove `brotherIN` è la funzione definita in sezione 6.2.3

A.7. Il vincolo di partizione

Il vincolo di partizione si ottiene combinando un vincolo di composizione con un vincolo di appartenenza disgiunta o quasi-disgiunta, come di seguito definito.

Vincolo di partizione

Definizione dei simboli:

Data una classe Y con attributo geometrico f ed una classe X con attributo geometrico g, il vincolo partizionato da Y verso X è definito come segue:

Sintassi:

```
vincolo Y.f partizionato X.g
```

Template OCL:

```
partizionato(Y, f, X, g) ≡
  dj-IN(X, g, Y, f) and
  ComposedOfConstraint(Y, f, X, g)
```

Vincolo di quasi-partizione

Definizione dei simboli:

Data una classe Y con attributo geometrico f ed una classe X con attributo geometrico g, il vincolo q-partizionato da Y verso X è definito come segue:

Sintassi:

```
vincolo Y.f q-partizionato X.g
```

Template OCL:

```
q-partizionato(X, g, Y, f)
  qdj-IN(X, g, Y, f) and
  ComposedOfConstraint(Y, f, X, g)
```

Anche per i vincoli di partizione sono applicabili alcune delle varianti già introdotte per i vincoli topologici, in particolare è possibile aggiungere selezioni e riferire il vincolo alla frontiera o alla proiezione planare del valore geometrico. Infine è possibile riferire il vincolo alla geometria di un attributo a tratti, a eventi o a sottoaree e legarli ad associazioni.

A.8. Vincoli di composizione con più classi vincolanti

Per definire formalmente la semantica di questo tipo di vincolo, è necessario specificare nuovi template OCL come segue (la funzione union si riferisce all'unione degli oggetti dei diversi tipi geometrici mentre la gUnion si riferisce all'unione degli insiemi di punti delle geometrie degli oggetti geometrici):

Vincolo compostoDa MultiVincolante

Definizione dei simboli:

Data una classe Y con attributo geometrico f ed un insieme di classi X_1, …, X_n con attributo geometrico g_1, …, g_n, il vincolo compostoDa da Y verso X_1, …, X_n si definisce nel seguente modo:

Sintassi:

vincolo Y.f *compostoDa* ($X_1.g_1$, …, $X_n.g_n$)

Template OCL:

$ComposedOfConstraint^{MULTI}(Y, f, X_1, g_1, \ldots, X_n, g_n)$

```
context Y
inv: self.f.Equals
     ( X1.allInstances.g1 ->
                 union(X2.allInstances.g2 ->
                 …
                 union(Xn.allInstances.gn) ->
      select(a:GU_Object| self.f.Contains(a)
                          or self.f.Equals(a)) ->
       iterate(b:GU_Object, acc: GU_Object = ∅ |
                            acc.gUnion(b))
     )
```

Anche per questi vincoli la definizione sintattica e il template OCL del vincolo vanno modificati qualora uno o entrambi gli attributi geometrici f e g siano di tipo GU_C*SurfaceB3D seguendo l'approccio descritto in precedenza.

La variante su associazione e tutte le altre combinazioni sono ottenibili per ovvia combinazione delle espressioni OCL presentate. Si noti che ai vincoli di composizione multivincolanti si applicano le stesse regole descritte nel vincolo di unione multivincolante per la determinazione dell'insieme di oggetti vincolanti selezionati per ogni oggetto vincolato.

Vincolo compostoDa Multi-vincolanti (variante con selezioni)

Definizione dei simboli:

Data una classe Y con attributo geometrico f ed un insieme di classi X_1, …, X_n con attributo geometrico g_1, …, g_n, il vincolo di composizione multi-vincolanti con selezioni si definisce nel seguente modo:

Sintassi:

vincolo $(\sigma_1(Y))Y.f$ *compostoDa*
$((\sigma_{2,1}(Y,X_1))X_1.g_1, \ldots, (\sigma_{2,n}(y,X_n))X_n.g_n)$

Template OCL:

$\text{ComposedOfConstraintMulti}^{SEL}(Y, f, X_1, g_1, \ldots, X_n, g_n)$

```
context Y
inv: σ0(self) implies
     self.f.Equals(
       X1.allInstances->select(x1:X1| σ2,1(Y,X1)).g1 ->
       union(X2.allInstances->select(x2:X2|
                                σ2,2(Y,X2)).g2 ->
      …
       union(Xn.allInstances->select(xn:Xn|
                                σ2,n(Y,XN)).gn))->
     select(a:GU_Object|
                    self.f.boundary().Contains(a)
                 or self.f.boundary().Equals(a))->
        iterate(b:GU_Object, acc: GU_Object = ∅ |
             acc.gUnion(b)))
```

Vincolo CompostoDa Multi-vincolanti su attributi a tratti con selezioni (casi TR/TR, GEO/TR e TR/GEO)

Definizione dei simboli:

Data una classe Y con attributo geometrico f e un attributo a tratti di nome a ed un insieme di classi X_1, …, X_n con un attributo geometrico g_i ciascuna e un attributo a tratti b_i ciascuna, il vincolo di composizione multi-vincolanti varianti TR/TR, GEO/TR e TR/GEO con selezioni si definisce nel seguente modo:

TR/TR

Sintassi:

vincolo $(\sigma_1(Y))Y.TrattiDi\_a(\sigma_{1a}(Y,a))$ *compostoDa*
$((\sigma_{2,1}(Y,X_1))X_1.TrattiDi\_b_1(\sigma_{2b,1}(Y,X_1,b_1)), \ldots,$
$(\sigma_{2,n}(Y,X_n))X_n.TrattiDi\_b_n(\sigma_{2b,n}(Y,X_n,b_n))$

Template OCL:

$ComposedOfConstraintMulti^{SEL}_{TR/TR}(Y, \sigma_1(Y), a, \sigma_{1a}(Y,a),$
$X_1, \sigma_{2,1}(Y,X_1), b_1, \sigma_{2b,1}(Y,X_1,b_1), \ldots,$
$X_n, \sigma_{2,n}(Y,X_n), b_n, \sigma_{2b,n}(Y,X_n,b_n))$

```
context Y
inv: σ1(self) implies
self.TrattiDi_a("σ1a(Y,a)")->
forall(t:GU_Object| t.Equals(
 X1.allInstances->
 select(x1:X1|σ2,1(self,x1)).
          TrattiDi_b1("σ2b,1(Y,X1,b1)")->
 union(…)->
 …
 union(Xn.allInstances->
 select(xn:Xn|σ2,n(self,xn)).
          TrattiDi_bn("σ2b,n(Y,Xn,bn)"))->
  select(a:GU_Object|t.Contains(a)or t.Equals(a) )->
  iterate(b:GU_Object, acc: GU_Object = ∅ |
                       acc.gUnion(b)))
)
```

GEO/TR

Sintassi:

vincolo $(\sigma_1(Y))Y.f$ *compostoDa*
$((\sigma_{2,1}(Y,X_1))X_1.TrattiDi\_b_1(\sigma_{2b,1}(Y, X_1,b_1)), \ldots,$
$(\sigma_{2,n}(Y,X_n))X_n.TrattiDi\_b_n(\sigma_{2b,n}(Y, X_n,b_n))$

Template OCL:

$ComposedOfConstraintMulti^{SEL}_{TR/TR}(Y, \sigma_1(Y), f,$
$X_1, \sigma_{2,1}(Y,X_1), b_1, \sigma_{2b,1}(Y,X_1,b_1), \ldots,$
$X_n, \sigma_{2,n}(Y,X_n), b_n, \sigma_{2b,n}(Y,X_n,b_n))$

```
context Y
inv: σ1(self) implies
self.f.Equals(
 X1.allInstances->
 select(x1:X1|σ2,1(self,X1)).
                      TrattiDi_b1("σ2b,1(Y,X1,b1)")->
 union(…)->
 …
 union(Xn.allInstances->
 select(xn:Xn|σ2,n(self,xn)).
                      TrattiDi_bn("σ2b,n(Y,X1,bn)"))->
 select(a:GU_Object| self.f.Contains(a)
                     or self.f.Equals(a))->
 iterate(b:GU_Object, acc: GU_Object = ∅ |
                      acc.gUnion(b))
)
```

TR/GEO

Sintassi:

vincolo $(\sigma_1(Y))Y.TrattiDi\_a(\sigma_{1a}(Y,a))$ *compostoDa*
$((\sigma_{2,1}(Y,X_1))X_1.g_1, \ldots, (\sigma_{2,n}(Y,X_n))X_n.g_n$

Template OCL:

$ComposedOfConstraintMulti^{SEL}_{TR/TR}(Y, \sigma_1(Y), a, \sigma_{1a}(Y,a), X_1, \sigma_{2,1}(Y,X_1), g_1, \ldots, X_n, \sigma_{2,n}(Y,X_n), g_n)$

```
context Y
inv: σ1(self) implies
self.TrattiDi_a("σ1a(Y,a)")->
  forall(t:GU_Object| t.Equals(
   X1.allInstances->select(x1:X1|σ2,1(self,x1)).g1->
   union(…)->
   …
  union(Xn.allInstances->
                    select(xn:Xn|σ2,n(self,xn)).gn)->
  select(a:GU_Object| t.Contains(a)or t.Equals(a))->
     iterate(b:GU_Object, acc: GU_Object = ∅ |
                              acc.gUnion(b)))
  )
```

Vincolo CompostoDa Multi-vincolanti (variante BND-PLN)

Definizione dei simboli:

Data una classe Y con attributo geometrico f ed un insieme di classi X_1, …, X_n con attributo geometrico g_1, …, g_n, il vincolo di composizione multi-vincolanti sulla frontiera (o sulla proiezione) si definisce nel seguente modo:

Sintassi:

vincolo Y.f.*BND* *compostoDa* ($X_1.g_1$, …, $X_n.g_n$)

Template OCL:

ComposedOfConstraintMulti^{B-}(Y, f, X_1, g_1, …, X_n, g_n)

```
context Y
inv: self.f.boundary().Equals(X1.allInstances.g1->
          union(X2.allInstances.g2)->
          …
          union(Xn.allInstances.gn)->
     select(a:GU_Object|
            self.f.boundary().Contains(a) or
            self.f.boundary().Equals(a))->
             iterate(b:GU_Object, acc: GU_Object=∅|
                     acc.gUnion(b))
     )
```

Sintassi:

vincolo Y.f *compostoDa* ($X_1.g_1.PLN$, …, $X_n.g_n.PLN$)

Template OCL:

ComposedOfConstraintMulti^{-P}(Y, f, X_1, g_1, …, X_n, g_n)

```
context Y
inv: self.f.Equals(X1.allInstances.g1.planar()->
          union(X2.allInstances.g2.planar())->
          …
          union(Xn.allInstances.gn.planar())->
     select(a:GU_Object| self.f.Contains(a)
                         or self.f.Equals(a))->
     iterate(b:GU_Object, acc: GU_Object = ∅ |
             acc.gUnion(b))
```

Similmente si definiscono le varianti con la funzione `boundary()` o `planar()` applicate sugli attributi g_i e `f` rispettivamente.