Reglas de traducción de restricciones entre OCL y LN

Autores:

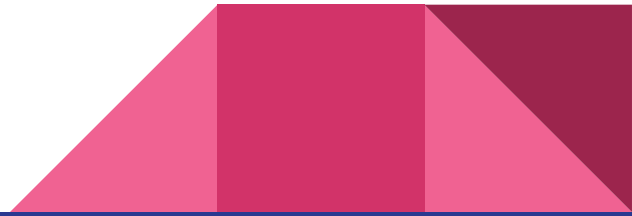
Lopez, Danae Claudia
Ibarguengoytia, Maria Amalia

Directora:

Pons, Claudia

Agenda

- Objetivos
- Motivación
- Marco teórico
- Herramienta
- Trabajos Relacionados
- Conclusiones
- Trabajos Futuros

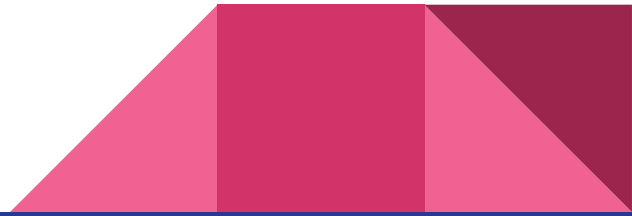


Objetivos

- Afianzar y resumir conceptos básicos de MDD (*Desarrollo de software Dirigido por Modelos*).
- Investigación sobre el lenguaje formal OCL y el lenguaje natural castellano.
- Análisis de factibilidad de automatización de la traducción.
- Prototipar una herramienta para la traducción automática.

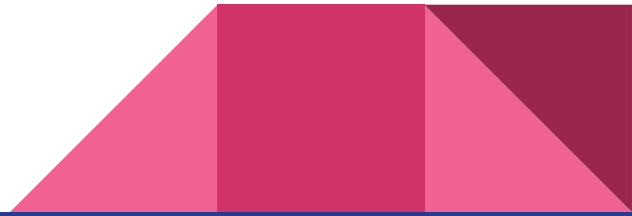
Motivación

- Modelos gráficos falta de expresividad.
- Modelos complementados.
- OCL: formal y tipado.
- OCL desventajas:
  - Difícil de entender y de escribir por personas no técnicas.
  - Tareas extras y manuales.
  - Personal especializado.



Marco Teórico

- MDD: Conceptos básicos.
- Lenguajes.
- Lenguaje Formal OCL.
- Xtext.
- ATL.

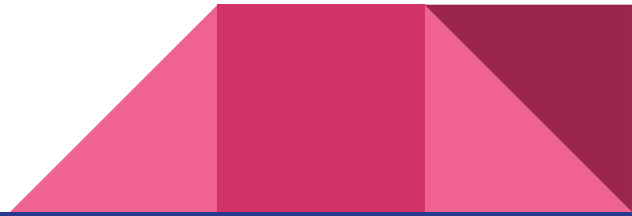


MDD - Conceptos básicos

- Es un paradigma de desarrollo de software centrado en modelos.
- Puntos claves:
  - *Abstracción*.
  - *Automatización*.
  - *Uso de estándares*.
- Metamodelos, Modelos, Metalenguajes.

MDD - Conceptos básicos

- Tipos modelos:
  - CIM: *Modelo Independiente de la Computación*.
  - PIM: *Modelo Independiente de la Plataforma*.
  - PSM: *Modelo Específico de la plataforma*.
  - IM: *Modelo de Implementación*.
- Tipos de Transformación:
  - M2M: *Modelo a Modelo*.
  - M2T: *Modelo a Texto*.



Lenguajes

- Sistema de comunicación compuesto por símbolos y sonidos.
- Clasificación:
  - Lenguaje Natural ⇒ Español, Inglés, Francés, entre otros.
  - Lenguaje Formal ⇒ Lenguajes de programación.
- Gramáticas - Libres de contexto:
  - BNF.
  - EBNF.
- Lenguaje Natural limitado o reducido.

Lenguaje Formal OCL

- Permite definir restricciones sobre los objetos de un modelo:
  - Restricciones adicionales al modelo gráfico → ***Modelo Conceptual***
  - Pre y post condiciones → ***Modelo de Operaciones***
- Las restricciones no generan efectos colaterales ni afectan el estado de un objeto.
- Modelos más precisos libres de ambigüedades.

Lenguaje OCL - Estructura de las aserciones

- **Contexto**: define donde la aserción es válida.
- **Un invariante**: la aserción en sí misma.

"El nombre del Hotel no puede ser ' ' ."

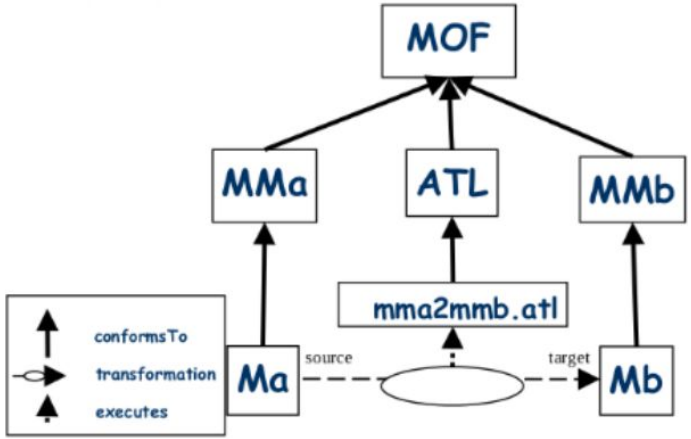
```
context Hotel
  inv: self.nombre<>''
```

Xtext

- Permite crear lenguajes específicos del dominio.
- Sintaxis similar a EBNF.
- Generar la gramática para obtener el metamodelo del lenguaje.
- Provee herramientas necesarias:
  - Compilador.
  - Interprete.
  - Editor.

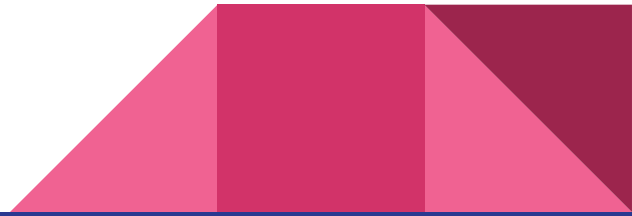
ATL

- Lenguaje de transformación de modelos.
- Híbrido: Declarativo e imperativo.
- Unidireccional.



ATL

- Module: contenedor del conjunto de reglas de transformación definidas.
- Header: Parte del module donde declara el nombre de los modelos de entrada y salida:
  - Modelo/os origen de solo lectura.
  - Modelo/os destino de solo escritura.
- Modo de ejecución:
  - Normal (From).
  - Refinado (Refine).

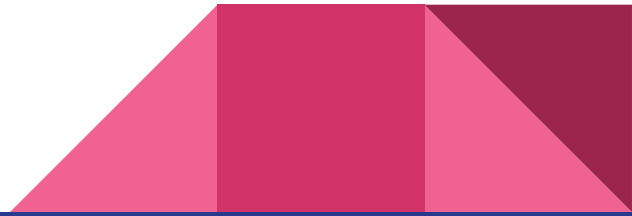


Herramienta - Diseño

- Procesamiento de OCL → Complete OCL / Pivot.
- Transformación de modelos → M2M / ATL.
- Representación de Lenguaje Natural → Gramática reducida / Xtext.

Herramienta - Implementación

- Gramática de Lenguaje Natural Reducido.
- Reglas de traducción ATL.

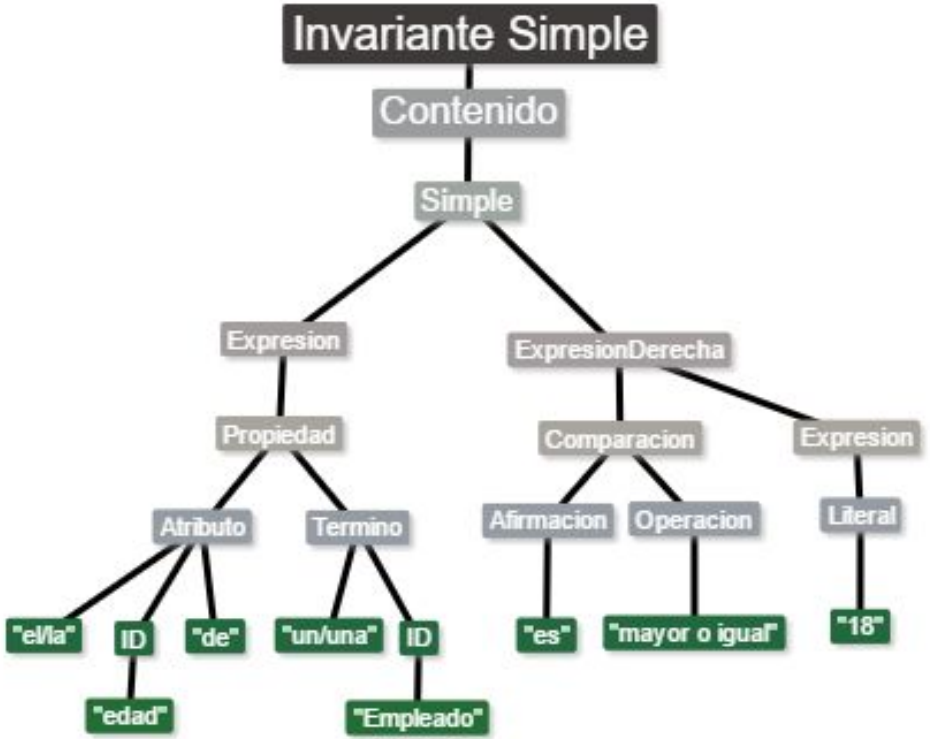


Invariante Simple

```
context Empleado
  inv Simple: self.edad >= 18
```

- Contexto Empleado inv(ariante) Simple yo mismo edad mayor o igual a 18:
  - No es sintáctica ni semánticamente correcta.
  - Difícil de entender.
- `el/la edad de un/una Empleado es mayor o igual a "18" .`
  - Respeta sintaxis y semántica.
  - Simple de entender.
- Extracción de componentes abstractos:
  - ‘el/la’ <Identificador> ‘de’ ‘un/una’ <Identificador> ‘es’ <Operacion> ‘18’.
  - Atributo, Termino, Comparacion, Literal
  - Niveles más abstractos => Simples, Compuestas e Iteraciones.

AST Invariante Simple



Invariante Compuesta

```
context Socio
  inv Compuesta:
    self.direccion <> '' and self.nombre <> '' and self.apellido <> ''
```

- Incluye operadores lógicos para conectar invariantes simples. Igual que en LN.
- Recursión para permitir oraciones compuestas por n oraciones simples.
- Contenido = Simple {Composicion}, Composicion = Nexo Contenido.

```
el/la direccion de un/una Socio es distinto de ""
y
el/la nombre de un/una Socio es distinto de ""
y
el/la apellido de un/una Socio es distinto de "" .
```

Invariante operación size

```
context Libro
    inv Size: self.copias->size() >= 1
```

- El resultado de size no es booleano por lo que siempre va acompañado de una comparacion.
- Reutilizar definición de la oración simple + Prefijo “cantidad de”.

```
el/la cantidad de copias de un/una Libro es mayor o igual a "1" .
```

Invariante operación colección Empty

```
context Biblioteca
    inv NotEmpty: self.libros->notEmpty()
```

- Resultado operacion es booleano, no requiere comparacion.
- Posibles traducciones:
  - “la colección de libros de una biblioteca (no) está vacía” => “colecciones” técnico.
  - “la cantidad de libros de una biblioteca (es mayor que) es igual a 0”.
- No requiere cambios en la gramática, se añade la comparacion.

```
el/la cantidad de libros de un/una Biblioteca es mayor que "0" .
```

Invariante con iteradores

- Se pueden presentar diferentes casos:
  - Único iterador que retorna un resultado booleano (exists, forAll).
  - Iterador que toma como entrada la salida del iterador previo (select(cond)->forAll(cond)).
  - Iterador que retorna una colección, esto requiere una comparación u operacion booleana (select).
- Cambia el inicio de la oración simple para algunos iteradores:
  - “el/la” pasa a ser “los/las” porque los nombres de las colecciones se expresan en plural.
  - exists => “entre los/las”.
  - forAll => “todos los/las”.
- Cambia la definición de propiedades para incluir las condiciones:
  - exists => “existe uno tal que”.
  - forAll => “satisfacen que”.
  - select => “tal que”.

Invariante iterador salida booleana - Exists

```
context Biblioteca
    inv exist: self.empleados->exists(e| e.rol = 'Bibliotecario/a')
```

- Exists(Condiciones): verdadero si **al menos 1** elemento es verdadero al evaluar condiciones.
- No tiene expresion derecha (comparacion + expresion):
  - Solo tiene expresión (Propiedad = **Atributo** + Término + **Iteración**).
  - Iteración contiene “existe uno tal que” + Contenido (Simple {Composición}) para condiciones.

```
entre los/las empleados de un/una Biblioteca existe uno/una tal que
el/la rol de un/una Empleado es igual a "Bibliotecario/a" .
```

Invariante iterador salida booleana forAll

```
context Autor
    inv forAll: self.obras->forAll(o| o.fechaDeEdicion >= self.fechaDeLaPrimerPublicacion)
```

- forAll(condiciones): Retorna verdadero si **todos** los elementos de la colección retornan verdadero al evaluar las condiciones.
- Traducción mediante la aplicación de las mismas reglas que para exists:
  - se aplica el prefijo “todos los/las” al inicio de la oración.
  - se define el prefijo de la iteración como “satisfacen que” seguido por las condiciones.

```
todos los/las obras de un/una Autor satisfacen que
el/la fechaDeEdicion de un/una Libro es mayor o igual a el/la fechaDeLaPrimerPublicacion de un/una Autor .
```

Invariante iterador salida colección Select

```
context Libro
    inv Select_notEmpty: self.copias->select(c| c.estadoDeLaCopia = 'Bueno')->notEmpty()
```

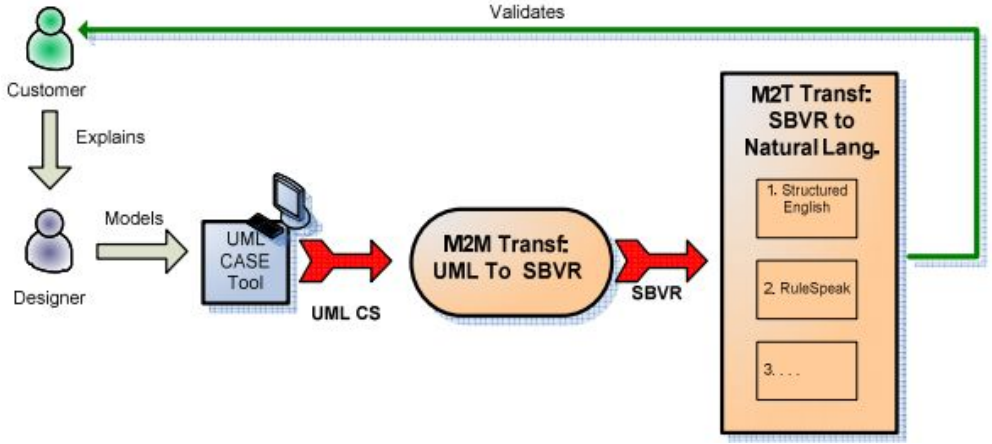
- Debe incluir una operacion o comparacion que retorna un booleano.
- Traducción similar a la de los iteradores exists, forAll:
  - prefijo oración depende de la operacion booleana que lo acompañe.
  - prefijo iteración se define como “tal que”.

```
el/la cantidad de copias de un/una Libro tal que
el/la estadoDeLaCopia de un/una Copia es igual a "Bueno" es mayor que "0" .
```

Herramienta - Video

Trabajos Relacionados

- UML/OCL a especificaciones SBVR: Transformación desafiante:
  - El objetivo es reducir la brecha entre expertos en el dominio y diseñadores.
  - Realiza 2 Transformaciones: UML/OCL a SBVR (M2M) y SBVR a Lenguaje Natural (M2T).
  - El lenguaje natural utilizado es inglés.

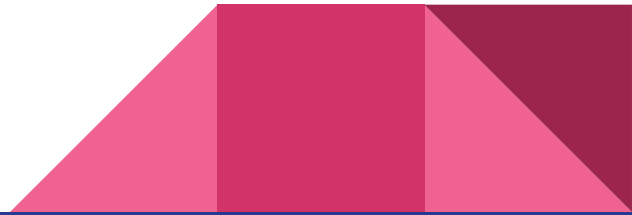


Trabajos Relacionados

- Generación de restricciones OCL a partir de una especificación en LN:
  - El objetivo es mejorar la usabilidad de OCL mediante la definición de restricciones en LN y traduciendolas automáticamente a lenguaje OCL.
  - Realiza 2 Transformaciones: LN a SBVR (T2M) y SBVR a OCL (M2M).
  - El lenguaje natural utilizado es inglés.

Trabajos Relacionados

- De especificaciones de software de lenguaje natural a modelos de clase UML:
  - El objetivo es mejorar la calidad de los modelos UML generados desde lenguaje natural.
  - Realiza una transformación LN a SBVR (T2M) y mapea los conceptos en SBVR con los objetos UML mediante el uso de código.
  - El lenguaje natural es inglés.



Trabajos Relacionados

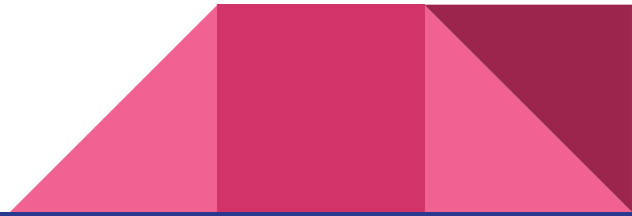
- Similitudes con nuestra herramienta:
  - Uso de MDD y Transformación modelos.
  - Objetivos incrementar la usabilidad OCL y reducir la brecha entre diseñadores y expertos.
- Diferencias con nuestra herramienta:
  - Orientadas a UML por lo que no permiten el uso de Ecore.
  - Se basan en el uso de lenguaje natural inglés.
  - Utilizan SBVR como lenguaje intermedio.

Conclusiones

- Aumentamos la usabilidad de OCL.
- Utilizamos CompleteOCL.
- Traducción al castellano.
- Demostramos que es posible realizar esta traducción mediante el uso de MDD y ATL.

Trabajos Futuros

- Extender la implementación.
- Detectar la repetición de contexto en las transformación.
- Definir género y número en la transformación.
- Generar las reglas de traducción ATL en sentido inverso.



Preguntas

¡Muchas Gracias!