Prototipo strumento semantico per la trasformazione da formati proprietari verso lo standard UBL (Report 4.1.3)

Sommario: questo documento è stato realizzato nell'ambito dell'attività OR 4.1

Autore: M. Vincini, M. Orsini, S. Bergamaschi, L. Sgaravato

Stato		Finale
File:		
Data:		19/05/2011 22.51
Diffusione	X	solo tra partner
		tra partner e partecipanti focus group
		pubblica

Introduzione

Il presente documento descrive, all'interno del laboratorio LISEA, le attività volte alla realizzazione di un prototipo per la trasformazione semantica di dati strutturati e proprietari nel formato standard UBL.

Lo standard UBL per la definizione del mercato elettronico

Nel mondo del Business, quando si parla di fatturazione e di documenti contabili informatici, si deve fare sempre riferimento al formato XML, raccomandato dal W3C già dal 1998 per lo scambio di messaggi nel commercio elettronico. Nel corso degli anni il formato XML ha però "causato" una serie di problemi, visto la presenza di numerosi formati-schemi diversi per la generazione di documenti. Il problema di questi diversi formati è stato ed è tutt'ora un problema di non poco conto, perchè per ogni scambio nasce un difetto di interoperabilità per la gestione e memorizzazione di diversi documenti con contenuti e formati tutti differenti tra loro. L' UBL, Universal Business Language risolve questo problema, identificando un formato unico per i documenti di business. L'UBL è stato creato e studiato dal OASIS (Advancing open standards for the global information society) ed è ad oggi utilizzato in moltissimi Paesi Europei, come ad esempio la Svizzera, la Danimarca, la Svezia e la Norvegia, per tutti gli scambi commerciali elettronici ed anche in contesti bancari.

L'UBL, derivando dall'XML, ovviamente non differisce da quest'ultimo. Come per l'XML, nell'UBL vengono definite diverse librerie di schemi per l' approvazione del documento creato e al fine di validare alcuni concetti-tag chiave di un documento (ad esempio fattura) contabile. Quindi abbiamo : il tag riferito alla denominazione sociale, al pagamento, all'indirizzo, alla Partita Iva, alla tipologia della fattura, all'ordine, al metodo di pagamento e così via. La definizione di specifici tag per il mondo del Business, offre una serie di vantaggi maggiori rispetto al "puro XML" , tra i quali :

- Minori costi di integrazione, sia all'interno che all'esterno delle imprese, attraverso il riutilizzo di strutture di dati comuni.
- Riduzione del costo di software commerciale.
- Una curva di apprendimento più facile.
- Minori costi di ingresso e quindi più rapida adozione da parte delle piccole e medie imprese (PMI).
- Formati standardizzati.
- Facilità di integrazione in diversi sistemi utilizzati.
- Standard per l'immissione dei dati economici.
- Integrazione con i vari sistemi EDI utilizzati delle imprese.

L' UBL è stato progettato per fornire una sintassi universalmente riconosciuta per gli scambi commerciali e per tutti i documenti di business legalmente vincolanti e di operare all'interno di un quadro commerciale standard, quale l' ISO 15000 (ebXML) che fornisce una completa infrastruttura "informatica" per il mondo del business. Dunque l'UBL permette anche di validare e gestire correttamente l'intero ciclo ordine-fattura, ma non solo. L'UBL infatti definisce correttamente anche il contenuto e la forma di un documento informatico contabile. E non è poco!!!. Scambiarsi documenti elettronici che hanno tutti una stessa caratteristica, è sicuramente un vantaggio per le singole aziende o Pubbliche Amministrazioni che possono scambiarsi le informazioni contabili in modo certo, sicuro, interoperabile e condivisibile con tutti i sistemi informativi. Nell'immagine seguente, viene descritto il modello di riferimento di una fattura UBL.

Al fine di rispondere alle diverse tipologie documentali del mondo del Business, l'UBL mette a disposizione una serie di schemi predisposti per la validazione di un documento XML contabile. Tra le altre ricordiamo :

Ordine
Risposta d'Ordine
Risposta d'Ordine Semplice
Modifica di un Ordine
Cancellazione di un Ordine
Avviso di Spedizione
Ricezione Pagamento

Fattura

Dunque l'UBL per mezzo di modelli specifici e schemi di diversi documenti contabili, definisce una vera e propria semantica per lo scambio elettronico commerciale. Non si tratta semplicemente di un linguaggio, ma molto di più. Lo standard UBL offre ulteriori 30 documenti elettronici normalizzati, tutti validi legalmente e concatenabili tra loro, mediante cui ingegnerizzare i processi di supply chain e di regolamento amministrativo interno all'azienda.
Pensiamo inoltre all'interoperabilità di tale formato e all' indipendenza dei diversi documenti informatici creati con lo standard UBL. In questo modo, anche se dovessero cambiare le regole del commercio elettronico o anche le regole per la generazione, gestione e conservazione delle fatture elettroniche, si potrebbe sempre e comunque modificare la sintassi UBL in modo del tutto automatico ed "indolore" e senza dipendere da un'altra organizzazione. Anche per le software house rendere-creare una piattaforma conforme all'UBL sarebbe a costo zero, sia in fase di sviluppo sia in fase di aggiornamento della normativa.

Generazione di un Wrapper XML-UBL

I documenti dello standard UBL, definiti attraverso l'XML Schema (formato xsd), possono essere interpretati secondo la semantica definita dall'OASIS e tradotti in un'ontologia utile a sistemi di integrazione dati per il mapping tra dati provenienti da sistemi aziendali.
La nostra proposta propone di utilizzare il sistema MOMIS (Mediator envirOnment for Multiple Information Sources) sviluppato presso il DII dell'Università di Modena e Reggio Emilia secondo un'architettura a mediatore per l'integrazione intelligente di informazioni estratte da sorgenti eterogenee.
A tal fine è stato realizzato un wrapper, uno strumento software per la traduzione degli schemi ubl-xsd verso il formato odli3 del sistema MOMIS.

Nelle prossime sezioni vengono presentati i punti principali del metodo di traduzione del wrapper.

Namespace e schemi composti
XML Schema fa uso dei namespace XML. Nel linguaggio ODLI3 il concetto di Namespace non è presente, per cui questo tipo di informazione non può essere mantenuta. Tuttavia è necessario fare attenzione al caso in cui un documento XML Schema faccia riferimento a oggetti (denizioni di tipi, dichiarazioni di elementi o attributi) appartenenti a namespace diversi sia dal target namespace dello stesso schema, sia dal namespace base del linguaggio (http://www.w3.org/2001/XMLSchema).
In questo caso infatti si potrebbero trovare nello stesso schema oggetti con lo stesso nome non qualificato, cioè privo del prefisso indicante il namespace, ma appartenenti a namespace diversi. Ad esempio si possono avere due Complex Type di nome po:Address e po:Address corrispondenti a definizioni diverse; nella traduzione si avrebbero così due interface con lo stesso nome Address, e si verificherebbe perciò una collisione nei nomi delle classi. Per ovviare a questo problema si utilizza il prefisso che in XML Schema identifica il namespace, anche per il nome delle Interface in ODLI3. Pertanto, l'esempio precedente porterebbe ad avere due interfacce di nome rispettivamente po_Address e po_Address; in questo modo non si creano collisioni in quanto XML Schema assicura l'unicità del nome all'interno del namespace. Un altro problema da affrontare nella traduzione riguarda la gestione degli schemi composti da più documenti, ed in particolare delle direttive include, redefine e import. Nel primo caso sarà necessario recuperare il file in cui è memorizzato lo schema che viene incluso ed estrarre da esso tutte le dichiarazioni e definizioni per poterle tradurre ed includere nel documento ODLI3 di destinazione la loro rappresentazione. Nel secondo caso si procederà come nel primo per tutti gli elementi che non sono ridefiniti, mentre per quelli ridefiniti sarà necessario sostituire la dichiarazione importata con quella ridefinita. Nel terzo caso, infine, si prederanno dal namespace importato le definizioni degli elementi che sono effettivamente referenziate nel corpo dello schema, seguendo le regole sui prefissi dei nomi descritte in precedenza.
Quindi, ogni volta che, nella traduzione di un costrutto, si incontrerà un riferimento ad un oggetto del namespace importato, si dovrà recuperare la definizione di tale oggetto, tradurla con le regole che andremo ad illustrare, ed inserirla nel documento ODLI3 finale.

Regole per la definizione dell'Algoritmo
A livello globale, la traduzione di un documento XML Schema verrà effettuata nel seguente modo:

1. per prima cosa sarà necessario includere/importare tutte le definizioni provenienti da namespace esterni nell'insieme delle strutture da tradurre;

2. in secondo luogo andremo a recuperare tutte le definizioni dei Simple Type globali (cioè non dichiarati in modo anonimo) e tradurli
3. quindi passeremo ad analizzare gli oggetti Element Declaration globali alla ricerca di eventuali gruppi di sostituzione;
4. a questo punto sarà affrontata la traduzione degli oggetti Attribute Group Definition e Model Group Definition.
5. infine andremo a tradurre tutti i Complex Type definiti globalmente;
6. come ultima cosa, sarà necessario prevedere una fase di post-processing in cui vengano validati i tipi complessi ODLI3 ed associate le definizioni agli identificatori usati nei tipi degli attributi; questa fase non è necessaria da un punto di vista logico, ma è richiesta dal punto di vista implementativo in quanto durante la costruzione dello schema ODLI3 l'associazione tra l'identificatore del tipo-classe e la sua definizione non è automatica, ma viene lasciata in sospeso, e deve quindi essere risolta al termine della creazione dello schema.

Lo schema finale delle regole ottenute è presentato nellla seguente tabella:

Nome	Costrutto XML Schema	Traduzione in ODL/3
Schemi su più documenti	<import schemaLocation=. . . /> <include schemaLocation=. . . /> <rede_ne schemaLocation=. . . />	Gestione definizioni esterne
Attribute	<xs:complexType name="[Cname]"> <xs:attribute name="[Aname]" type="[Atype]"/> </xs:complexType>	interface [Cname] { attribute [Atype] [Cname]_[Aname]?; }
Attribute required	<xs:complexType name="[Cname]"> <xs:attribute name="[Aname]" type="[Atype]" use="required"/> </xs:complexType>	interface [Cname] { attribute [Atype] [Cname]_[Aname]; }
Attribute con valore fixed	<xs:complexType name="[Cname]"> <xs:attribute name="[Aname]" type="[Atype]" fixed="400"/>	interface [Cname] { const [Atype] [Cname]_[Aname] = 400; }
Attribute reference	<xs:attribute name="[Aname]" type="[Atype]"/> <xs:attribute name="[Aname]" type="[Atype]"/> </xs:complexType>	interface [Cname] { attribute [Atype] [Cname]_[Aname]; }
Element	<xs:complexType name="[Cname]"> <xs:element name="[Aname]" type="[Etype]"/> </xs:complexType>	interface [Cname] { attribute [Etype] [Ename]; }
Element reference	<xs:element name="[Ename]" type="[Etype]"/> <xs:complexType name="[Cname]"> <xs:attribute ref= "[Ename]"	interface [Cname] { attribute set<[Etype]> [Ename]?; }

	minOccurs= ”0” maxOccurs=” unbounded”/>	
Element con dichiarazione di tipo anonimo	<xs:complexType name=”[Cname]”> <xs:element name=”[Ename”> <xs:complexType> <!- . . . - > </xs:complexType> </xs:element> </xs:complexType>	interface [Cname] { attribute [Cname_Ename_type] [Ename]; } interface [Cname_Ename_type] { . . . }
Element opzionale	<xs:complexType name=”[Cname]”> <xs:element name=”[Aname]” type=”[Etype]” minOccurs= “0”/> </xs:complexType>	interface [Cname] { attribute [Etype] [Ename]?; }
Element con cardinalità (1,n)	<xs:complexType name=”[Cname]”> <xs:element name=”[Aname]” type=”[Etype]” maxOccurs= “unbounded”/> </xs:complexType>	interface [Cname] { attribute set<[Etype]> [Ename]; }
Element tradotto con array	<xs:complexType name=”[Cname]”> <xs:element name=”[Aname]” type=”[Etype]” minOccurs= “4” maxOccurs=”4”/> </xs:complexType>	interface [Cname] { attribute [Etype] [Ename][4]; }
Element con cardinalità (0,5)	<xs:complexType name=”[Cname]”> <xs:element name=”[Aname]” type=”[Etype]” minOccurs= “2” maxOccurs=”5”/> </xs:complexType>	interface [Cname] { attribute set<[Etype]> [Ename]?; }
nillable Element	<xs:complexType name=”[Cname]”> <xs:element name=”[Aname]” type=”[Etype]” nillable= “true”/> </xs:complexType>	interface [Cname] { attribute [Etype] [Ename]?; }
Substitution Group	<xs:element name=□[Sname]□ type=”[Stype]”/> <xs:element name=”[SubName1]” type=”[Stype]” substitutionGroup=”[Sname]”/> <xs:element name=”[SubName2]” type=”[Stype]” substitutionGroup=”[Sname]”/> <xs:complexType name=”[Cname]”> ... <xs:element ref=”[Sname]”/> ...	interface [Sname]_union { attribute [Stype] [Sname]; } union { attribute [Stype] [SubName1]; } union { attribute [Stype] [SubName2]; } interface [Cname] { ... attribute [Sname]_union [Sname];

	</xs:complexType>	... }

Substitution Group con Abstract Element	<xs:element name="[Sname]" type="[Stype]" abstract= "true"/>
	<xs:element name="[SubName1]" type="[Stype]" substitutionGroup="[Sname]"/>
	<xs:element name="[SubName2]" type="[Stype]" substitutionGroup="[Sname]"/>
	<xs:complexType name="[Cname]">
	...
	<xs:element ref="[Sname]"/>
	...
	</xs:complexType>
		interface [Sname]_union
		{ attribute [Stype] [SubName1];
		}
		union
		{ attribute [Stype] [SubName2];
		}
		interface [Cname]
		{
		...
		attribute [Sname]_union [Sname];
		...
		}

Complex Type Generico	<xs:complexType name="[Cname]">
	<xs:sequence>
	<xs:element name=name type=xs:string/>
	<xs:element name=street type=xs:string/>
	<xs:element name=city type=xs:string/>
	. . .
	</xs:sequence>
	</xs:complexType>
		interface [Cname]
		{
		attribute string name;
		attribute string street;
		attribute string city;
		}

Complex Type a contenuto Semplice derivato da un Simple Type	<xs:complexType name="[Cname]">
	<xs:simpleContent>
	<xs:extension base="[SimpleTypeName]">
	<xs:attribute name="[Aname]" type="[Atype]"/>
	</xs:extension>
	</xs:simpleContent>
	</xs:complexType>
		interface [Cname]
		{
		attribute [SimpleTypeName] [Cname]_node;
		attribute [Atype] [Cname]_[Aname]?;
		}

Complex Type a contenuto Complesso derivato da un altro Complex Type	<xs:complexType name="[BaseCname]"> <xs:sequence> . . . </xs:sequence> </xs:complexType> <xs:complexType name="[DerivedCname]"> <xs:complexContent> <xs:extension base="[BaseCname]"> . . . </xs:extension> </xs:complexContent>	interface [BaseCname] { . . . } interface [DerivedCname] : [BaseCname] { . . . }

	</xs:complexType>	

Complex Type con contenuto nullo	<xs:complexType
	name="[EmptyCname]">
	<xs:complexContent>
	<xs:restriction
	base="xs:anyType">
	<xs:attribute name="[Aname]"
	type="[Atype]"/>
	</xs:restriction>
	</xs:complexContent>
	</xs:complexType>
		interface [EmptyCname]
		{
		attribute [Atype]
		[EmptyCname]_[Aname] ?;
		}

Simple Type di tipo lista	<xs:simpleType name="[ListTypeName]"> <xs:list itemType="[itemTypeName]"/> </xs:simpleType>	typedef list<[itemTypeName]> [ListTypeName];

Simple Type di tipo unione	<xs:simpleType
	name="[UnionTypeName]">
	<xs:union
	memberTypes="[memberType_1]
	[memberType_2] . . . "/>
	</xs:simpleType>
		union [UnionTypeName]
		switch(val) {
		case 1: [memberType_1];
		case 2: [memberType_2];
		case . . .
		}

Simple Type derivato da un tipo lista con sfaccettatura length	<xs:simpleType name="[ListName]"> <xs:list itemType="[itemType]"/> </xs:simpleType> <xs:simpleType name="[DerivedName]'> <xs:restriction base="[ListName]"> <xs:length value="[i]"/> </xs:restriction> </xs:simpleType>	typedef list<[itemType]> [ListName] ; typedef array<[itemType],[i]> [DerivedName] ;
Simple Type enumerato	<xs:simpleType name="[EnumType]"> <xs:restriction base="[BaseType]"> <xs:enumeration value="[Val_1]"/> <xs:enumeration value="[Val_2]"/> <xs:enumeration value="[Val_3]"/> . . . </xs:restriction> </xs:simpleType>	enum [EnumType] { [Val_1], [Val_2], [Val_3], . . . }
Simple Type traducibile con un tipo range	<xs:simpleType name="[Rname]"> <xs:restriction base="[IntegerType]"> <xs:minInclusive value="[min]"/> <xs:maxInclusive value="[max]"/> </xs:restriction>	typedef range [min],[max] [Rname] ;

	</xs:simpleType>	

Attribute Group Definition	<xs:schema ...>
	<xs:attributeGroup
	name="[AGname]">
	...
	</xs:attributeGroup>
	<xs:complexType
	name="[Cname]">
	...
	<xs:attribute ref="[AGname]"/>
	</xs:complexType>
	</schema>
		interface [AGname]_group
		{ ... }
		interface [Cname]
		{
		...
		attribute [AGname]_group
		[AGname];
		}

Model Group Definition	<xs:schema ... > <xs:group name="MGname"> . . . </xs:group> <xs:complexType name="Cname"> <xs:group ref="MGname"/> . . . </xs:complexType> </xs:schema>	interface [MGname]_mgroup { . . . } interface Cname { . . . attribute [MGname]_mgroup [MGname]; }
Identityconstraint unique	<unique name = "[Uname]"> <selector xpath = "[XPathS]" /> <field xpath = "[XPathF]" </unique>	Tradotto con un costrutto key. L'espressione XPathS serve per determinare l'interface in cui inserire la chiave. L'espressione XPathF determina gli attributi da includere nella chiave.
Identityconstraint key	<key name = "[Kname]"> <selector xpath = "[XPathS]'/> <field xpath = "[XPathF]" </key>	Tradotto con un costrutto key. L'espressione XPathS serve per determinare l'interface in cui inserire la chiave. L'espressione XPathF determina gli attributi da includere nella chiave.
Identityconstraint keyref	<keyref name = "[Kname]" refer = '[Rname]"> <selector xpath = "[XPathS]"/> <field xpath = "[XPathF]" </keyref>	Tradotto con un costrutto foreign_key. L'espressione XPathS serve per determinare l'interface in cui inserire la chiave. L'espressione XPathF determina gli attributi da includere nella chiave. L'attributo refer di valore [Rname] serve a determinare la chiave cui riferire la nuova foreig_key
Annotation	<xs:complexType name="[Cname]"> <xs:annotation> <xs:documentation>	interface Cname { const string annotation_1 ". . . documentation . . . ";

	. . . documentation . . . . . . </xs:documentation> <xs:appinfo> . . . appinfo . . . . . . </xs:appinfo> . . . </xs:annotation>	. . . }

Model Group <all>	<xs:complexType
	name="[Gname]">
	<xs:all>
	<xs:element name="[Ename_1]"
	type=
	"[Etype_1]"/>
	<xs:element name='[Ename_2]"
	type= "[Etype_2]"/>
	</xs:all>
	</xs:complexType>
		interface [Gname]
		{
		attribute [Etype_1] [Ename_1];
		attribute [Etype_2] [Ename_2];
		}

Model Group Sequence	<xs:complexType
	name="[Gname]">
	<xs:sequence>
	<xs:element name="[Ename_1]"
	type= "[Etype_1]"/>
	<xs:element name="[Ename_2]"
	type="[Etype_2]"/>
	</xs:sequence>
	</xs:complexType>
		interface [Gname]
		{
		attribute [Etype_1] [Ename_1];
		attribute [Etype_2] [Ename_2];
		}

Model Group choice	<xs:complexType name="[Gname]"> <xs:choice> <xs:element ref="[Union_1]"/> <xs:element ref="[Union_2]"/> . . . </xs:choice> </xs:complexType>	interface [Gname] { attribue [Gname]_choice [Gname]_choice; } interface [Gname]_choice { attribute ... [Union_1]; } union { attribute ... [Union_2]; } union { attribute ... }

Modifiche dinamiche di una Ontologia

La gestione di grandi quantità di informazioni è di crescente importanza nelle realtà odierna, con una sempre più massiccia presenza di grandi organizzazioni; difficoltà che sono state affrontate introducendo il concetto di ontologia. In questo contesto, l'operazione di integrazione di informazioni e creazione di una ontologia richiede ancora molti sforzi da parte del progettista spesso combinato all'impiego di tecniche di costruzione ad hoc, poiché esistono pochi approcci metodologici che affrontano la situazione a posteriori. In ogni caso, anche una ontologia in origine completa e priva di errori, dopo un determinato periodo di tempo richiede aggiornamenti per rimanere coerente con i cambiamenti del mondo reale. Possono essere un esempio i mutamenti non prevedibili dell'ambiente in cui l'ontologia opera, oppure, l'aggiunta o la modifica dei requisiti degli utenti rispetto a quelli iniziali su cui era stata costruita. Questo discorso diventa ancor più presente se si parla di applicazioni che operano in ambiente di Internet e del Web Semantico, basate su sorgenti dati eterogenee fortemente distribuite.

Dalla definizione di Gruber, universalmente riconosciuta come la più completa, si possono individuare tre aree di interesse dove è possibile agire nel contesto di dinamica di una ontologia:

- cambiamenti nel dominio: riguardano cambiamenti nell'ambiente in cui l'ontologia è attiva, esempio modifiche sulla struttura dei modelli del database selle sorgenti, come il merge di due sorgenti.
- cambiamenti nelle concettualizzazioni condivise: un aspetto notevole da rimarcare è che questo tipo di cambiamenti può presentarsi più volte. Fensel [16] descrive le ontologie come reti dinamiche di significati, nelle quali il consenso è ottenuto in un processo sociale di scambio di informazioni e di significato. Differenti scopi possono necessitare differenti visioni sul dominio e conseguentemente una differente concettualizzazioni. Questa visione da un doppio ruolo alle ontologie nello scambio di informazioni: sia come prerequisito per lo scambio di informazioni che il risultato di questo processo di scambio. Per esempio quando una ontologia è adattata per un nuovo scopo o dominio, le modifiche rappresentano cambiamenti nella concettualizzazione. La trasformazione rimane quindi limitata al modo in cui la concettualizzazione è formalmente memorizzata, generando un cambiamento di facile soluzione, poiché dovrebbe rimanere nella semantica, come le varianti della specificazione dovrebbero essere equivalenti e causare solo cambiamenti sintattici.
- cambiamenti nelle specificazioni: non sempre lo scopo per il quale una applicazione viene implementata rimane lo stesso nel tempo. Esso si può estendere ed evolversi in base, ad esempio, alle esigenze degli utenti che lo utilizzano, in questo caso può risultare necessario anche modificare l'ontologia di riferimento.

In letteratura si possono trovare due correnti di pensiero riguardanti le tecniche da adottare per affrontare la tematica: l'approccio basato sulla evoluzione e l'approccio basato sulle versioni. Queste tipologie di approccio sono oggetto di studio anche in altri campi della gestione della dinamica delle informazioni, quali database relazionali e altre applicazioni software.

Approccio basato sulla evoluzione

L'approccio che si basa sulla evoluzione dell'ontologia affronta il problema della dinamica in tutta la sua complessità, gestendo ogni conseguenza che un cambiamento dell'ontologia impone sulle sorgente ad esso collegate. Questo è un approccio abbastanza complesso basato principalmente sul concetto di "consistenza di una ontologia". Heflin, nella tesi di dottorato, si avvale di esempi e di un insieme di principi per definire quali sono le basi di questo approccio:
"The revision of an ontology should not change the well-formedness of resources that commit to an earlier version of the ontology."

In pratica una cancellazione effettiva probabilmente accade raramente, più facilmente un termine è rimosso perché può essere fuso con un altro termine o viene preferito un nome diverso.
"Resources that commit to a revised ontology can be integrated with resources that commit to compatible prior versions of the ontology."
Una modifica in una parte dell'ontologia può generare delle inconsistenze sia in un'altra porzione della stessa ontologia, che nelle applicazioni che sono basate su quella ontologia.

Da qui deriva la seguente definizione:

Data un linguaggio di logica del primo ordine Γ, una ontologia è una quintupla BPEAV,,,,, dove il vocabolario

V ⊂ Sp, cioè l'insieme dei predicati inclusi nei simboli non logici di Γ, gli assiomi A ⊂ W, cioè l'insieme infinito

di formule well-formed che possono essere costruite in Γ, E ⊂ OE ,cioè l'insieme delle ontologie estese di O,

P⊂OE, è l'insieme delle versioni precedenti dell'ontologia, e B ⊂ P è l'insieme delle ontologie con cui O è backwards-compatible.

Un altro problema, estremamente rilevante nel campo del web semantico, quando si parla di evoluzione di ontologie, è quello della "scalabilità di una ontologia". L'utilizzo della logica del primo ordine non permette l'implementazione di questa caratteristica, è necessario quindi ricercare altre vie.
Un aspetto della scalabilità è quello di implementare algoritmi di ragionamento a risorse limitate (che si limitano nel tempo di computazione, nel il numero di step o altro) dove lo scopo della ricerca non è la necessità di ottenere tutte le risposte complete al problema ma solo un insieme di risposte corrette. Data l'estensione del Web, non sembra possibile che ogni ragionatore abbia accesso a tutte le asserzioni, come del resto è improbabile che un algoritmo ben formato possa essere realmente completo in senso globale.
Un altro aspetto della scalabilità è quello di ridurre l'espressività del linguaggio. Questa è stata un'importante direzione di sviluppo per la comunità della rappresentazione della conoscenza che ha provato a specificare la complessità computazionale dei linguaggi con varie funzioni, al fine di sviluppare linguaggi che massimizzino l'espressività ma allo stesso tempo minimizzino la loro complessità.
La struttura maggiormente nota che si basa su un approccio evolutivo per la gestione degli aspetti dinamici è il framework KAON, articolato in sei fasi successive, ha una struttura circolare, dal momento che la validazione dei cambiamenti realizzati può automaticamente indurre nuovi cambiamenti per ottenere la consistenza del modello o per soddisfare le aspettative degli utenti.

Un studio, su quali siano le trasformazioni possibili applicabili ad una ontologia, è stato effettuato nel progetto realizzato da Leehneer. Partendo da un approccio simile a quello precedentemente descritto, esso sfrutta la definizione di semantica dei mondi possibili14, sviluppata da Saul Kripke nel 1963, per definire quale possa esser l'evoluzione di una ontologia.
Il modello ha una struttura formata da tre componenti: un set K di mondi possibili, un insieme di relazioni di accessibilità R(u,v) definite su K e un funzione di evoluzione Φ (Ω, φ) in modo tale da rendere una ontologia formale come un insieme di oggetti matematici.
In questa analogia, il passaggio da una ontologia Ω1 ad un'altra Ω2, si riduce ad una processo di evoluzione formato da una trasformazione, definita come una sequenza finita di operazioni di modifica, atomiche e complesse, che permettono di passare da Ω1 ad Ω2. Obiettivo di questo progetto è quello di inserire all'interno dell'ontologia nuove informazioni che non entrino in conflitto con quello contenute nella ontologia di partenza.
Dalla teoria AGM si possono individuare tre tipi di operazioni applicabili nella trasformazione di una ontologia:

- Espansione: un nuovo elemento di conoscenza φ viene inserito all'interno dell'ontologia Ω con la gestione della propagazione di tutte le informazioni aggiuntive ad essa collegate, non piccole nel caso di ontologie di grandi dimensioni.
- Revisione: un nuovo elemento di conoscenza φ che risulta essere inconsistente per l'ontologia Ω viene aggiunto, questo porta all'eliminazione di alcune vecchie decisioni interne ad Ω per mantenere la consistenza dell'ontologia risultante.
- Contrazione: alcuni elementi di conoscenza φ di Ω vengono ritrattati per dare spazio a nuovi elementi di conoscenza.

Le trasformazioni possibili nello spazio delle ontologie possono essere meglio visualizzati avvalendosi della griglia di Lindenbaum dove ogni nodo rappresenta una possibile ontologia e ogni collegamento una trasformazione tra due di esse.
Queste tre non sono, tuttavia le uniche trasformazioni che si possono avere se si parla di evoluzione di una ontologia. Il progetto di Leenheer ha esaminato altri tre tipi di relazione:

- trasformazione per preservare le equivalenze;
- propagazione dei cambiamenti a cascata;
- contenuti possibili o necessari.

Trasformazione per preservare le equivalenze

La relazione di equivalenza è una trasformazione riflessiva, simmetrica e transitiva che genera una classificazione delle ontologie possibili in un set di classi disgiunte tra loro, creando in questo modo un insieme di rappresentazioni equivalenti della stessa concettualizzazione.

Propagazione dei cambiamenti a cascata
Se una ontologia Ω1 è inclusa in un'altra ontologia Ω 2, una trasformazione T1 da Ω1 a Ω*1 crea un procedimento a cascata sulle ontologie che includono Ω1, generando T2 tra Ω2 e Ω*2. Se la catena di inclusione continuasse fino a Ωn, la trasformazione andrebbe avanti generando trasformazioni fino a Tn fra Ωn e Ω*n. La propagazione dei cambiamenti è monotona verso l'alto, cioè una trasformazione a metà della catena di inclusioni tra ontologie non comporta trasformazioni in quelle interne.

Contenuti possibili o necessari
Gli elementi di informazione φ, appartenenti ad un' ontologia secondo una relazione $\Phi(\varphi,\Omega) \rightarrow \{T,F\}$ sul Ω, si possono classificare tra i contenuti necessari o possibili. Questa divisione usa come parametro di classificazione il valore di verità che l'elemento assume nelle ontologie Ω0, accessibili dalla corrente ontologia attraverso una trasformazione consistente. Un elemento si definisce necessario se è vero in tutte le ontologie, si definisce possibile se è vero solo in alcune di queste. In questo caso l'inserimento di tale elemento può essere semplificata, senza che l'ontologia perda consistenza.

Approccio basato sulle versioni

L'approccio basato sulle versioni, a differenza del primo basato sulle evoluzioni, gestisce i cambiamenti delle ontologia creando e definendo differenti versioni della stessa, ciò comporta la presenza di un numero elevato di copie della stessa ontologia. Nasce quindi la necessità di sviluppare algoritmi che consentano al progettista di riconoscere e distinguere le diverse versioni a sua disposizione per garantire la realizzazione e il mantenimento delle ontologie. Come nell'approccio basato sull'evoluzione, la propagazione dei cambiamenti necessari all'ottenimento della nuova versione dovrebbe essere esaminata in dettaglio anche se la nuova versione dell'ontologia ottenuta con il l'approccio qui esaminato si può considerare a tutti gli effetti come una nuova ontologia, che può anche non avere dei punti di contatto con la versione precedente. Questo approccio è molto vantaggioso nel caso di ontologie ad uso distribuito, poiché agevola notevolmente la fase di aggiornamento dell'ontologie, dando la possibilità ad applicazioni che non necessitano dell'implementazione del cambiamento di continuare a lavorare con la precedente versione.
Quando si parla di relazioni tra diverse versioni di una ontologie bisogna esaminare tre aspetti:

- Differenziare le versioni delle relazioni dalle concettualizzazioni delle relazioni interne all'ontologia;
- Discutere nel caso di aggiornamento dell'ontologia se ci sono delle discrepanze tra i cambiamenti nelle concettualizzazioni e quelli nelle specificazioni;
- Definire le vie per applicare questi aggiornamenti sull'ontologia, pacchetti di cambiamento, packaging of changes.

Mapping di ontologie

Il mapping di ontologie è diventato un punto chiave negli ultimi anni. Le motivazioni di questo interesse lo si può ricercare in ambito commerciale aziendale: fusione di società o riorganizzazioni comporta spesso la fusione dei propri sistemi informativi, sempre più spesso gestiti da ontologie, inoltre a volte si ricorre alla fusione di numerose ontologie per ottenerne una di migliore qualità. Tuttavia anche il merging a tempo di esecuzione può essere cruciale specialmente se si tiene conto del fatto che le ontologie sono diventate estremamente comuni nel WorldWideWeb dove forniscono semantica alle annotazioni presenti nelle pagine di internet. Come ampliamente discusso l'eterogeneità delle informazioni sul web possono portare a lavorare con differenti ontologie in domini molto simili, queste diversità devono coesistere con le interazioni tra i sistemi. Una scelta possibile per rendere compatibili le diversità è quella di stabilire regole di mappatura tra diverse ontologie e riunirle tutte a tempo di esecuzione.
Come conseguenza delle motivazioni appena espresse, sono nati numerosi tool per eseguire il mapping di ontologie, basati su due approcci principali collegati tra loro:

fusione (merge) → viene creata un'unica ontologia aderente alle informazioni contenute da quelle di partenza.
allineamento → vengono stabilite dei link di relazione tra le ontologie.

Allineamento di ontologie

In generale si parla di mapping di ontologie quando vengono create tra queste delle corrispondenze nel vocabolario e negli assiomi in esse contenuti, avvalendosi di strutture matematiche. L'allineamento di ontologie è una tipologia di mapping di ontologie, che mette in relazione una ontologia intermediaria, chiamata "Articulation ontology", con le sorgenti ontologiche da allineare.
L'allineamento crea quindi un livello superiore di legami di relazione, mantenendo separate le ontologie di base, che rimangono completamente autonome tra di loro sia durante che dopo il processo di allineamento. Questa tecnica viene solitamente applicata nel caso di ontologie con schemi del domino complementari con quelli del livello superiore. Per esempio, un specifico dominio di una ontologia può essere allineata ad una ontologia centrale in CyC stabilendo dei legami in CyC's upper- and middle- level di ongologie. Questi specifici domini che non possono essere integrati con la conoscenza di base di Cyc, rimangono separati dalle ontologie che contengono riferimenti a concetti di questo tipo. Questo è uno specifico esempio di allineamento di ontologie riguardante la creazione di collegamenti tra agenti che utilizzano differenti tipi di modelli di ontologie. Infatti una volta allineate le ontologie e determinato i concetti minimi necessari per creare una comunicazione sufficiente tra gli agenti (cioè le intersezioni delle concettualizzazioni nei modelli in oggetto ritenute necessarie) gli agenti possono interagire tra loro sfruttando queste proprietà.
Uno esempio di allineamento di due ontologie è quello eseguito dal Formal Concept Analysis (FCA) è descritto nell'articolo di De Sauza pubblicato dall'IEEE '04. Il progetto utilizza il thesaurus POSET per l'allineamento delle ontologie e crea una struttura a griglia, visibilmente immediata, per comparare e ricavare le sovrapposizioni, e quindi le differenze, dei due schemi. Per calcolare e valutare la similarità tra elementi dell'ontologia, si avvale di un algoritmo di soglia basato su due funzioni; la prima relativa alle caratteristiche comuni, e non, presenti nei termini del thesaurus, la seconda, invece, relativa alla struttura della rappresentazione a griglia dell'ontologia..
Un elenco su quali possono essere le tecniche adottate per stabilire relazioni tra concetti di ontologia, viene descritto nell'algoritmo di merge implementato nel progetto DOGMA. L'algoritmo richiama, al suo interno, una fase di allineamento tra ontologie, formalmente definita come l'interpretazione dei termini che formano la base lessicale di una ontologia sΩ rispetto a quelli di un'altra ontologia tΩ. Questo processo avviene attraverso due passaggi, in seguito descritti, che portano alla identificazioni dei legami di mapping e quindi all'allineamento delle ontologie in esame.

Trovare regioni di intersezione dei domini
Il primo passaggio necessario quando si parla di comparazione, è quello di specificare quali sono le parti delle ontologie che corrispondono alle intersezione dei loro rispettivi domini.

Trovare concetti equivalenti tra le ontologie
Questo passaggio estrae ed esamina le varie tipologie di eterogeneità che ci possono essere tra concetti ritenuti semanticamente equivalenti, cioè presenti in entrambe le ontologie ma indicati con termini differenti. Per ovviare al problema di una errata corrispondenza tra ontologie (a causa di errori di digitazione, ambiguità dei termini, utilizzo di differente terminologia per indicare lo stesso concetto, e così via) la fase di comparazione dovrebbe fare riferimento ai significati del concetto, invece che le etichette dei temimi che li descrivono. Il grado di similarità tra due concetti c1 e c2 è misurato attraverso funzioni che riducono il calcolo al confronto con i termini del linguaggio naturale. Viene qui di seguito proposta una lista delle tecniche per poter determinare il grado di similarità fra i termini delle ontologie, attraverso una comparazione con il linguaggio naturale. Tutte queste tecniche sono basate sulle differenze sintattiche fra i termini, non considerano il valore semantico, è quindi necessaria una successiva fase di analisi critica per l'interpretazione dei risultati.

- **Porter Stememer**. Implementato solitamente come parte del processo di normalizzazione nei sistemi di Information Retrieval, lo stemming rimuove le estensioni morfologiche e inflessioni poste alla fine dei termini. Un suo utilizzo, ad esempio, è quello di ridurre la forma plurale di una parola alla propria base singolare (papers → paper). Il simbolo → indica com'è la parola prima e dopo lo stemming.
- **Levenshtein Distance**. Questa misura, chiamata della "Edit Distance", indica la distanza di similarità fra due termini e , vista come numero di cancellazioni, inserimento o sostituzioni richiesti per trasformare in , dove maggiore è il valore risultante maggiore è la differenza tra i termini.

- **Longest Common Prefix/Suffix**. Il risultato di similarità fra 2 termini e , può essere calcolato come la lunghezza del prefisso o suffisso comune più lungo,
- **Longest Common Substring**. Questo metodo individua la sottostringa comune più lunga tra e , cioè l'insieme più lungo di caratteri in ordine che compare in e.
- **Metaphone Algorithm**. Questo metodo mette in relazione i termini su una base probabilistica di equivalenza. Il metaphone Algortihm riduce ogni stringa di input a caratteri in codice metafonici, utilizzando regole fonetiche relativamente semplici. Alcuni esempi: "university" e "universities" si trasformano in UNFRST e UNFRSTS, mentre due termini come "faculty" e "faculties" vengono associati allo stesso codice FKLT.

Identificare inter-relazioni tra ontologie

Per poter identificare inter-relazioni tra concetti di differenti ontologie bisogna prima di tutto definire l'esatta semantica di queste relazioni. Viene qui di seguito riportata una lista delle diverse relazioni con semantica predefinita:

- **SubClassOf**: questa relazione collega un concetto con domini comuni.
- **Generalize**: questa relazione generalizza in un nuovo concetto due concetti che hanno domini intersecati.
- **Part of** : questa relazione collega un concetto con una sua parte rappresentata da un altro concetto; questo tipo di relazione è tipica tra concetti con domini disgiunti.
- **IstanceOf** : questa relazione indica che un oggetto e' un istanza di un concetto.

Una metodologia che permetta di automatizzare la ricerca di inter-relazioni tra concetti di ontologie è chiamata SUMO (Suggested Upper Merged Ontology).

Merge di ontologie

Il merge di ontologie è applicabile nel caso di ontologie con domini sovrapposti o simili tra loro. Il risultato di questo processo è un'unica ontologia contenente tutte le informazioni di quelle iniziali. La fusione di più ontologie in una unica è spesso utilizzata con lo scopo di estenderne una in particolare tra queste. Consideriamo per esempio Unified Medical Language System (UMLS) e Galen Coding Reference model (CORE), se si unissero tutte queste differenti ontologie si creerebbe un'unica ontologia del dominio medico.
Un possibile algoritmo di integrazione di due ontologie, sperimentato nel progetto DOGMA e discusso nell'articolo di Meersman, si può articolare in diverse fasi per mettere in relazione i differenti componenti dell'ontologia, per trovare e risolvere i conflitti tra la rappresentazione e i concetti nel mondo reale, e per effettuare il merge tra ontologie conformi in una globale. In particolare il progetto in esame implementa questo algoritmo in quattro passaggi collegati:

Pre-integrazione delle ontologie

Nella fase di pre-integrazione delle ontologie vengono stabilite la politica generale di integrazione da adottare: quali ontologie integrare, come integrarle, scegliere le strategie e le tecniche da adottare, stabilire l'ordine di integrazione e se necessario assegnare preferenze all'intera ontologia o parte di questa. Questa fase è formata quindi da un insieme di scelte decisionali prese da un operatore umano sulla base della sua esperienza.

Comparazione delle ontologie
La comparazione delle ontologie effettua un'analisi che determina le correlazioni fra i concetti equivalenti delle ontologie, individua le inter-relazioni tra ontologie. La comparazione di ontologie è sinonimo di "allineamento di ontologie", rimandiamo quindi la discussione al paragrafo successivo.

Conformazione dell'allineamento

Nella fase che controlla la conformità dell'allineamento, viene accertato il corretto adempimento del parametro di consistenza dell'ontologia. Quando una relazione di allineamento viene proposta dal sistema o dal progettista, questa viene istantaneamente comparata con quelle già presenti per evitare di inserire dei conflitti. La validazione di questo inserimento deve controllare che non ci siano cicli tra gli concetti interni all'ontologia e che non ci siano conflitti derivanti da allineamenti di classi inferiori.

Merge dell'ontologia e ristrutturazione

In questa fase le sorgenti ontologiche vengono unite seguendo i modelli e le relazioni stabilite nelle fasi precedenti per creare una ontologia globale che chiameremo Ωmerge. Le operazioni che si possono individuare nella fase di merge tra ontologie sono principalmente tre:

- fusione di elementi di informazione equivalenti;
- specializzazione;
- generalizzazione.

Operazione di fusione

Questo tipo di operazione si applica tra concetti considerati equivalenti. Fondere due concetti richiede una prima fase per nominare, tra i due, quello di riferimento come nell'ontologia Cmerge Ωmerge, concetto contenente il risultato finale della fusione, ed una seconda per comparare le relazioni lessicali dei concetti equivalenti e, nel caso siano equivalenti, copiarle associandole a . Tuttavia, anche nel caso contrario di relazioni lessicali non lessicalmente equivalenti tra loro, in cui non ci siano inter-relazioni con altre dell'ontologia, queste vengono associate a Cmerge.

Operazione di specializzazione

L'operazione di specializzazione viene applicata quando durante la fase di allineamento nasce una relazione del tipo "c1 SubClassOf c2" i cui concetti appartengono ad ontologie differenti. L'interpretazione che si può dare è questa: sia data una relazione di eredità non-monotona, nella quale, quindi, si può modificare una certa proprietà ereditata.

Operazione di generalizzazione

L'operazione di generalizzazione tra due concetti c1 e c2 si applica quando si evidenzia la necessità di creare un nuovo concetto cg, tale per cui "c1 is_a cg" e "c2 is_a cg", si introduce, così, una generalizzazione dei concetti in un altro concetto, attraverso una operazione simile quella di fusione, prima descritta. A differenza di questa, tuttavia, la fusione è relativa solo alle relazioni equivalenti, che vengono associante al nuovo concetto cg, mentre i due concetti e c1 e c2 rimangono separati.

Due approcci al confronto

Dopo avere presentato, nei paragrafi precedenti, in dettaglio in cosa consistono i due approcci sviluppati per operare un'integrazione fra ontologie, si può ragionare su quanto siano intimamente collegati.
L'allineamento infatti, come abbiamo visto, è una fase molto importante della procedura di merge dove vengono ricavate, attraverso le varie misure introdotte, le similitudini fra differenti ontologie; si può quindi considerare il merge di ontologie come un'estensione del puro allineamento.
La differenza di utilizzo è, di conseguenza, conseguente alle esigenze specifiche dell'integrazione; quando è sufficiente semplicemente stabilire delle corrispondenze fra differenti ontologie l'allineamento è la soluzione preferibile, perchè creando un livello superiore non va a modificare la struttura delle ontologie esistenti. Il suo svantaggio, parlando da un punto di vista computazionale, è che mantenere molte di queste sovrastrutture rallenterebbe di molto l'elaborazione. In questo caso, quindi, è meglio utilizzare il merge, per ottenere una struttura dati più compatta e che permetta una più facile elaborazione.
Concludendo, i due approcci sono praticamente l'uno, il merge, l'estensione dell'altro, l'allineamento, e l'utilizzo dell'uno o dell'altro è una scelta demandata al progettista in base alla complessità del proprio specifico dominio di lavoro.

Per valutare le performance dell'algoritmo implementazione, si può eseguire, per entrambi gli approcci, un confronto con l'allineamento manuale delle stesse ontologie. I parametri utilizzanti sono quelli di "recall" e "precision", comunemente impiegati nei contesti di Information Retrieval.
Indicato con H l'insieme degli elementi simili secondo il giudizio ed esperienza dall'operatore, ed M l'insieme degli oggetti simili rilevati dal tool di integrazione, i due parametri si possono definire come segue:

- Il parametro **recall**, viene calcolato sui documenti rilevanti recuperati, e ne misura la percentuale rispetto al totale contenuto nella raccolta.
- Il parametro **precision** invece riguarda i documenti restituiti dalla ricerca e rappresenta la percentuale di documenti rilevanti.

Integrazione di due Global Virtual View

Linee guida: inserimento di una nuova sorgente

Il sistema MOMIS prevede una successione di passaggi per consentire all'utente di creare il proprio schema delle concettualizzazioni. Il primo passo è la fase di annotazione delle sorgenti e di creazione del Common Thesaurus, il secondo è la classificazione della conoscenza rappresentata e per concludere la creazione della vista globale unica e la sua annotazione.
Lo schema descritto definisce dei passaggi statici, unidirezionali ma nel momento in cui si presenti la necessità di gestire delle azioni di modifica ad uno schema già creato e salvato, questo schema non e' più applicabile. Esistono due metodologie per risolvere questa trasformazione:

- ripartire dal procedimento iniziale creando una nuova ontologia,
- sviluppare tecniche per apportare le modifiche partendo dallo schema finale da modificare.

Il primo approccio è di tipo statico, poiché non prevede una vera e propria modifica dell'ontologia, ma aggira il problema ricreando un nuovo schema. Questo procedimento, oltre a portare ad un maggior impiego di tempo, non conserva le informazione di mapping memorizzate poiché ricrea un cluster ex-novo, risulta, quindi, essere di poca utilità in vista di uno sviluppo di software industriale. Al contrario, la seconda soluzione preserva le informazioni già raccolte, come mapping e Common Thesaurus, in particolare quelle definite manualmente dall'utente. Questa ultima soluzione è stata adottata nello sviluppo di questa tesi e rientra nel campo della gestione delle dinamiche di una ontologia, in particolare è stato implementato un tool seguendo le tecniche proposte dall'approccio basato sulla evoluzione delle ontologie.
Il passaggio da una versione già strutturata di una ontologia, Ω1, ad un'altra, Ω2, può avvenire attraverso varie trasformazioni. Secondo la teoria AGM, è possibile creare dei raggruppamenti di queste operazioni in tre tipologie standard di modifica:

- Espansione o integrazione di una nuova sorgente: una nuova sorgente deve essere integrata all'interno della GVV, apportando nuove classi locali e nuovi attributi locali da mappare nello schema già formato.
- Contrazione o eliminazione di informazioni: una sorgente, o parte di questa, deve essere eliminata perché dichiarata obsoleta, cioè non riflette le necessità richieste nelle concettualizzazioni del mondo reale.
- Revisione o aggiornamento di sorgenti già presenti: una sorgente deve essere aggiornata nella sua struttura di classi e attributi, questo può essere visto come una combinazione delle due azioni precedentemente citate, cioè eliminazione del concetto da modificare ed inserimento del nuovo concetto modificato.

Nella sezione precedente è stato riportato come queste problematiche siano state trattate in letteratura e come siano state affrontate nei progetti di ricerca, soprattutto nell'ambito dell'Information Retrieval.
Nell'ambito specifico in questa tesi, si è voluto focalizzare l'attenzione sulla prima tipologia di modifica, con l'obiettivo, di sviluppare un algoritmo per l'integrazione di una o più sorgenti sfruttando quello che si può definire una comparazione o confronto di due versioni della stessa ontologia. La trattazione di questo algoritmo viene rimandata al paragrafo successivo.
Nella sezione relativa al merge di ontologie, si è già parlato del fatto che la fusione di più ontologie in una unica globale, nasca a volte dalla necessità di estendere particolari ontologie iniziali. In questa ottica è stato sviluppato un algoritmo per effettuare la fusione di due ontologie, creando una vista globale unica ΩMerge,

partendo da due ontologie indipendenti tra loro, ΩOld e ΩNew, anche se rappresentanti entrambe due versioni della stessa vista globale:

- ΩOld, è l'ontologia iniziale, creata sulle vecchie sorgenti, da integrare con i nuovi elementi di informazione;
- ΩNew, è una ontologia creata ex-novo, che contiene sia le informazioni delle nuove sorgenti che i concetti definiti "globali" relativi alla ontologia iniziale ΩOld, vista come unica sorgente locale.

Nella fase di pre-integrazione di due ontologie di questo tipo il primo problema che si pone è quale sia la "direzione" più opportuna da seguire per la creazione della ΩMerge: integrare ΩNew con le informazioni raccolte nella precedente versione, sostituendo, quindi, i concetti globali con le informazioni definite sulle sorgenti locali della vista iniziale, oppure aggiornare quest'ultima con i nuovi concetti raccolti. I due approcci sono ambivalenti, entrambi presentano, infatti, le stesse problematiche riguardante la comparazione delle schemi e, quindi, la ricerca delle relazioni tra concetti globali dell'ontologia iniziale con quelli locali della ΩNew. Nonostante ciò esiste una sostanziale differenza da un punto di vista computazionale: le primitive di trasformazioni richieste dal primo approccio risultano essere di maggiore complessità rispetto al secondo. La sostituzione di una informazione globale già presente richiede, infatti, la cancellazione e l'inserimento di diverse informazioni, mentre il secondo approccio risulta essere, da questo punto di vista, meno gravoso poiché implementa esclusivamente un algoritmo di espansione, cioè di inserimento di nuove informazioni. La scelta ricade quindi sulla tipologia delle applicazioni che utilizzano la struttura della ontologia.

Regole base per creare un algoritmo di comparazione

L'algoritmo di comparazione tra due versioni di una stessa ontologia deve tener conto dei seguenti presupposti:
a) ΩMerge non deve ridefinire una classificazione tra le sorgenti appartenenti alla ΩOld, ma deve tutelare e mantenere le informazioni raccolte relative al Common Thesaurus e alla Mapping Table con i concetti globali (classi e attributi).
b) l'integrazione di una nuova sorgente non deve perdere le informazioni relative alle struttura delle sorgenti locali della ΩOld: classi e attributi locali.
c) l'integrazione della nuova sorgente deve avvenire partendo dalla ΩOld, considerata come vista globale e non come insieme di sorgenti separate l'una dall'altra.
d) valutare i casi particolari che si possono presentare nell'inserimento di una nuova sorgente relativi alla modifica alla struttura dell'ontologia iniziale.
Questi presupposti non sono necessariamente scollegati tra di loro, si può affermare infatti che alcune siano la conseguenza di altre, esempio non si può pensare di rispettare la clausola (a), cioè di mantenere le informazioni raccolte nel Common Thesaurus della ΩOld, senza rispettare la clausola (b).

Ricerca delle inter-relazioni tra le due ontologie

Come già citato, gli approcci che si possono adottare per creare l'allineamento di due ontologie possono essere varie, sia di tipo semantico che sintattico, tuttavia quando si parla di integrazioni di due versioni della stessa ontologia per l'inserimento di una nuova sorgente le tecniche che si possono usare possono diventare più mirate. Si ha come punto di partenza che esistono già delle relazioni intrinseche tra schemi delle due ontologie, che collegano concetti definiti "globali" della ΩOld a concetti definiti "locali" della ΩNew, tali relazioni inter-schema esistono per il solo fatto che l'una, ΩNew , ingloba una vista globale della prima, ΩOld. I concetti in esame rappresentano, quindi, la stessa concettualizzazione anche se non sempre formalizzata in modo equivalente. Le relazioni tra questi concetti forniscono una base indispensabile per trovare i collegamenti inter-schema tra le due ontologie e quindi, per far migrare le informazioni secondo il verso stabilito. La ricerca di tali legami non è tuttavia banale, specialmente se viene modificato il nome con cui i concetti vengono definiti; questo problema è presente soprattutto nei progetti dove sono implementati algoritmi di cluster che non prevedono l'assegnamento di livelli di priorità alle sorgenti per la scelta del nome da dare alla classe globale. In questo caso i legami sintattici di similarità su concetti comuni vengono persi, risulta quindi utile seguire un approccio sintattico per effettuare la ricerca delle inter-relazioni. Una possibile alternativa è ricercare la provenienza di tale concetto definendo qual è la sorgente di informazione ad esso associata e, nel caso questa risulti essere la ΩOld, effettuare tutti gli aggiornamenti necessari per mantenere la consistenza dell'ontologia. Viceversa, se l'algoritmo di cluster dà la possibilità di mantenere una relazione

di omonimia tra concetti comuni, definiti nelle due ontologie, allora si possono applicare gli algoritmi di ricerca sintattica precedentemente scartati.

Modifica della struttura dello schema ontologico iniziale

La regola base (d) stabilisce un vincolo che conserva lo schema iniziale della ΩOld, cioè la propagazione dei cambiamenti, legati all'inserimento di nuovi concetti, non deve comportare una modifica alla struttura della ontologia iniziale. Ad esempio se si considerano due concetti globali c_glob1 e c_glob2 della ΩOld, questi possono essere definiti come il risultato di una classificazione di concetti locali.
L'inserimento di una nuovo concetto potrebbe portare all'eliminazione di uno dei due concetti globali, supponiamo c_glob1, e alla migrazione dei suoi collegamenti locali verso c_glob2. Accettare questo tipo di cambiamento a priori, cancellare quindi c_glob2 senza valutare quale siano le relazioni della ΩNew che hanno portato a questa modifica e, soprattutto, senza considerare le relazioni che esprimono i legami tra i concetti globali e quelli locali interni alla ΩOld, non è sempre corretto.
In queste situazioni le strade che si possono percorrere sono diverse e specifiche caso per caso, possono variare sia in base alla struttura dello schema dell'ontologia che al valore che questo attributo globale occupa all'interno di questo schema.

Algoritmo di Comparazione

Il processo di integrazione di una nuova sorgente non si conclude con la creazione della GVVnew, essa non può essere, infatti, accettata come schema finale, in quanto non rispetta i requisiti iniziali, perde ogni informazione riguardante le lsourceOld, il Common Thesaurus ed il Mapping Table con le classi e gli attributi globali della GVVold. Una volta che le due viste sono state create, si deve procedere alla loro integrazione in una vista globale unica. La GVVall può essere ottenuta in due modi: in avanti verso la GVVnew, oppure indietro verso la GVVold. Nel primo caso, che prevede una sostituzione delle informazioni contenute nella GVVnew relative alla vista precedente con quelle delle lsourceOld ad esse associati, l'aggiornamento e integrazione del Common Thesaurus e la Mapping Table, non si ha una modifica della GVVold e la GVVall viene creata da modifiche sulla GVVnew. Nel secondo caso, accade l'opposto, la GVVold viene incrementata dalle nuove informazioni raccolte. Nonostante i due approcci siano entrambi esatti, è sembrato più conveniente seguire il secondo. Le motivazioni sono principalmente di ordine computazionale, mentre il primo richiede una sostituzione degli attributi e delle classi e un'aggiunta delle vecchie informazioni relative alle relazioni delle vecchie sorgenti, il secondo prevede solo una allargamento della vecchia vista con le nuove informazioni.

Considerata questa composizione, si possono distinguere tre casi:

1. Una nuova classe globale si compone di una classe locale relativa ad una classe globale della vecchia GVV e di una o più classi locali della nuova sorgente;
2. Una nuova classe globale si compone unicamente di classi locali della nuova sorgente;
3. Una nuova classe globale si compone di più classi locali relative a classi globali della vecchia GVV e di una o più classi locali della nuova sorgente.

Caso 1: Una nuova classe globale contiene una corrispondenza a una classe globale della GVVold

Se un'applicazione basa il suo funzionamento sulla struttura della GVV, questo scenario non costituisce problemi, poichè non si ha una variazione dello schema della vecchia vista globale ma solo un suo incremento di informazioni. L'algoritmo ricerca nella lista delle classi globali della GVVold quella che a cui si riferische la gcOld della gcNew, e incrementa le sue classi locali con quelle della nuova sorgente mappate nella gcNew in studio. Nello specifico: esegue una scansione su tutte le classi globali della GVVold, memorizzate nell'algoritmo precedentemente definito, fino a che non trova la corrispondenza; a questo punto memorizza la corrispondenza nella variabile GlClassO e la estende con le classi locali della nuova sorgente. Le lcNew sono sia le classi locali esaminate precedentemente, che non hanno dato rapporto falso al controllo di corrispondenza, che quelle successive a quella in esame. Su queste ultime non viene svolto il controllo, perché ritenuto ridondante. Il controllo, viene sempre stabilito sulla base del nome della sorgente e non il nome della classe. Quando non c'è corrispondenza tra quest'ultimi, poiché l'algoritmo prevede di

estendere le informazioni verso la vecchia vista, viene mantenuto come nome quello di gcOld, cioè gcOldname.
Stessa cosa vale per gli attributi gcNewAtt, questi sono costituiti da un insieme di attributi in parte comuni alle vecchie classi globali e in parte generati dall'apporto delle nuove classi locali. L'aggiornamento viene fatto sugli attributi globali della classe globale nella vecchia GVV, gcOld, inoltre, non viene eseguito ad ogni inserimento, ma viene svolto una sola volta alla fine del ciclo sulla gcNew.

Caso 2: Una nuova classe globale si compone unicamente di classi locali della nuova sorgente
Si tratta del caso più semplice, in cui la variabile corrispondenza ha valore zero. Viene creata, nella GVVnew, una nuova classe globale gcNewcon nome gcNewName, che mappa una o più nuove classi locali; non vi sono quindi relazioni che uniscono classi locali della nuova sorgente con quelle della vecchia vista, GVVold.

Caso 3: Una nuova classe globale contiene più corrispondenze a classi globali della GVVold
Si tratta della situazione più critica in quanto la struttura della GVV è modificata pesantemente attraverso il processo di integrazione. La distinzione tra le classi viene modificata poiché più vecchie classi globali vengono agglomerate in una nuova classe globale.
Se un'applicazione appoggia il suo funzionamento sulla struttura della GVV, questo scenario potrebbe costituire dei problemi. Questa situazione si può riscontrare se una lcNew ha forti legami con più classi globali definite nello schema della GVVold. Questo caso ha richiesto una maggiore attenzione, in quanto le decisioni che si possono prendere variano a seconda dei casi, e in base al livello di libertà da dare all'utente.
Un primo approccio è quello di unire le classi come suggerito dal nuovo schema, questa soluzione è stata subito scartata, perché questo è esito di un cluster eseguito su un Common Thesaurus che non tiene conto delle relazioni tra lsourceOld e, quindi, delle lcOld, ma che mantiene solo delle relazioni limitate ai concetti globali della vecchia vista. Inoltre esso genera una modifica della struttura precedentemente formata, creando possibili disagi alle applicazioni.
Un altro approccio è quello di mantenere divise le gcOld e creare una nuova classe globale nella GVVold contenente le lcNew. Inizialmente l'algoritmo prevedeva questo tipo di soluzione, successivamente l'analisi e la sperimentazione hanno portato alla considerazione che questa soluzione può portare ad errori di errata concettualizzazione della realtà nello schema. A differenza di quanto potrebbe accadere in una relazione di similitudine, in questo caso le classi non sono scollegate tra loro ma sono unite da una relazione, quella di generalizzazione, che è indipendente da ciò che può restituire un'analisi delle relazioni con le lsourceOld.
Nell'algoritmo si è scelto, quindi, di adottare una ulteriore soluzione dando massima libertà all'utente. Le classi globali, a cui fanno riferimento le gcOld, non vengono modificate in modo tale da non alterare lo schema della GVVold senza tener conto delle relazioni generate dalle le sue sorgenti locali, mentre le lcNew non vengono mappate nel nuovo schema, ma vengono lasciate nelle mani del progettista, il quale deciderà in prima persona come comportarsi, in base alla sua esperienza.