POLITECNICO DI MILANO

Facoltà di Ingegneria dell'Informazione

Corso di Laurea Specialistica in Ingegneria Informatica

ONTOLOGY DESIGN PATTERN:
Analisi di pattern esistenti e alcune nuove proposte

Relatore: Prof. Marco Colombetti

Tesi di Laurea di:

Luca Bogani

matricola 725057

Anno Accademico 2010-2011

INDICE

1. Introduzione 3

2. I Design Pattern e l'informatica 5
2.1 I Design pattern e le ontologie 9

3. Le categorie di design pattern ontologici 13
3.1 ODP di Presentazione 15
3.2 ODP di Ragionamento 16
3.3 ODP Lessico-Sintattici 17
3.4 ODP di Corrispondenza 22
3.5 ODP Strutturali 30
3.6 ODP di Contenuto 36

4. ODP Logici 39

5. Nuove proposte di ODP Logici 55

6. Conclusioni 61

7. Bibliografia 63

Capitolo 1 – Introduzione

L'ingegneria della conoscenza e lo sviluppo delle ontologie stanno vivendo un periodo molto florido: l'avanzare della tecnologia e il progresso verso macchine sempre più "intelligenti" hanno come base una ontologia, spesso però limitata a un solo piccolo dominio, per prevenire l'esplosione della complessità di realizzazione della stessa.

La tesi svolta da me per il corso di laurea triennale aveva cominciato ad esplorare le possibilità che i design pattern, strumenti ben noti nel mondo della progettazione e dell'ingegneria del software, potevano offrire al mondo affine della progettazione e sviluppo delle ontologie. Oggigiorno i design pattern stanno venendo presi in considerazione seriamente come supporti adatti, se non necessari, per la creazione di nuove basi di conoscenza.

Il progetto NeOn, nato nel 2006 con finanziamenti a livello europeo, aveva lo scopo di creare mezzi informatici e una metodologia precisa che aiutassero lo creazione di nuove applicazioni semantiche, per garantire delle soluzioni economiche in grado di aiutare gli sviluppatori attraverso i cambiamenti che il mondo del software stava allora attraversando, a causa dell'enorme quantità di informazioni che si stavano rendendo disponibili tramite Internet e che potevano essere sfruttate e combinate tra loro pur provenendo da fonti molto diverse. Una costola di quel progetto, il sito

www.ontologydesignpatterns.org, aveva lo scopo preciso di costituire la prima raccolta online di design pattern dedicati alle ontologie. Ora il sito è indipendente dal progetto NeOn, e grazie alla sua struttura a "wiki", ogni ricercatore o appassionato può contribuire alla creazione di nuovi pattern o alla valutazione di quelli già proposti. Il sito ha la dichiarata ambizione di diventare un'autorità nel campo delle ontologie, con un comitato di analisti che valuterà e certificherà i design pattern in grado di superare gli standard richiesti, creando così il primo repository ufficiale e certificato di design pattern.

Si è quindi scelto di strutturare il lavoro in 2 fasi: la prima di analisi e descrizione della situazione attuale della ricerca su questi design pattern, analizzando sia i concetti di base (design pattern e ontologie *in primis*) sia i lavori già svolti e presenti sul sito in questione; la seconda fase di sviluppo di nuovi pattern e pubblicazione sul sito.

L'elaborato seguirà questa struttura, con un primo capitolo dedicato alla spiegazione introduttiva per i profani ai concetti di design pattern e di ontologia, e ai cenni storici riguardanti le prime interazioni fra i due ambiti. Il capitolo successivo si occuperà approfonditamente del lavoro svolto sul sito succitato, dando una panoramica sulle categorie finora individuate dal comitato di studiosi e mostrando esempi e altre particolarità legate a ciascuna categoria, con un capitolo interamente dedicato agli ODP Logici.

Il quinto capitolo si occuperà infine dei pattern che si sono elaborati per la pubblicazione e loro eventuale valutazione da parte del comitato di qualità.

Come di consueto il testo si concluderà con una riflessione conclusiva sul lavoro svolto, e con indicazioni di possibili sviluppi futuri di questo ambito di ricerca.

Capitolo 2 – I Design pattern e l'informatica

Come molti ormai già sapranno, i design pattern non hanno origine nell'informatica, nonostante la loro stupefacente utilità in questo campo.

Fecero la loro prima apparizione su un libro di architettura, "A Pattern Language", pubblicato da Christopher Alexander nel 1977. L'architetto, assieme ai suoi collaboratori, illustrava nel libro ben 253 schemi ricorrenti volti a dare linee guida per altri architetti su come creare una città più funzionale e a misura d'uomo, ma anche esteticamente piacevole. Certo degli schemi ripetuti e riutilizzabili non erano un concetto nuovo: infatti, Alexander sviluppò l'idea dei design pattern studiando le città medievali europee cercando di scoprire quali elementi comuni avessero tra loro; lo stesso concetto di soluzione ricorrente per un problema altrettanto ricorrente non era sicuramente sconosciuta ad artigiani e costruttori di qualunque tipo: in fondo, fin dai tempi delle chiese romaniche e gotiche gli schemi costruttivi si ripetevano simili in diversi parti del mondo. Il concetto di base era noto perfino ai romani, i quali ogni volta che fondavano una nuova città partivano dallo stesso schema ortogonale di cardo e decumano.

La novità degli schemi proposti stava quindi nella loro codifica precisa e standardizzata, che però lasciava comunque ampi margini di movimento a chiunque avesse deciso di ispirarsi a uno degli schemi presenti nel libro.

Questa apparentemente strana commistione di libero arbitrio e vincoli dati al progettista sono sicuramente il punto di forza dei pattern: mentre su alcuni aspetti ritenuti fondamentali ogni pattern si mostra piuttosto rigido e immutabile, su aspetti secondari viene lasciata totale libertà. Due esempi di questo contrasto possono essere due dei pattern proposti da Alexander nel suo libro: "A place to wait" e "Window Place", dai nomi molto intuitivi. Il primo è un pattern dedicato alla realizzazione di sale d'attesa, siano esse di qualunque genere: dalla sala d'attesa di un grande ospedale al preingresso di uno studio privato di un notaio fino alla sala d'attesa in una stazione ferroviaria. Il pattern dà indicazioni su dove posizionare le sedie, il bancone della biglietteria o la scrivania della segretaria e, se presenti, i servizi igienici per la clientela; gli altri dettagli sono lasciati da scegliere al progettista: se mettere sedie o poltrone o divanetti, i materiali delle stesse, e i colori e l'eventuale arredamento aggiuntivo, sempre che non entri in contrasto con il mobilio regolamentato dal pattern. In maniera analoga il pattern "Window place" dà indicazioni al progettista su come piazzare le fonti di luce naturale all'interno di una stanza, basandosi sulla posizione degli ingressi nella stanza e sulla funzione della stessa; il materiale degli infissi e il loro colore, ma anche la forma e la dimensione (entro certi limiti) della finestra sono lasciati completamente alla volontà del progettista.

Un altro grande vantaggio dei design pattern è la loro modularità: essendo così specifici su alcuni aspetti e completamente generici su altri, è possibile combinare due o più pattern senza che essi vengano a dare indicazioni contrastanti l'uno con l'altro. I due esempi appena mostrati infatti possono essere applicati entrambi per la creazione di una sala d'aspetto ospedaliera senza cadere in contraddizione o contrasto fra loro.

Questi punti di forza sono gli stessi motivi per cui l'applicazione dei design pattern alla programmazione ha avuto un grande successo. Dopo dieci anni dalla pubblicazione del libro di Alexander, infatti, una pubblicazione presentata alla conferenza OOPSLA del 1987 (Object Oriented Programming, System, Languages and Applications) da Kent Beck e Ward Cunningham, mostrava al mondo informatico i potenziali vantaggi che i design pattern avrebbero apportato al mondo dell'ingegneria del software. Da questa pubblicazione presero lo spunto per il loro libro l'ormai famosa "Gang of four", composta Gamma, Helm, Johnson e Vlissides.

Nel 1994, infatti, venne pubblicato il libro *Design Patterns: Elements of Reusable Object-Oriented*, in cui i design pattern venivano ufficialmente e in maniera concreta applicati al mondo della programmazione e dell'ingegneria del software. Il libro è ancora oggi un punto di riferimento per i programmatori di tutto il mondo, e arrivato ormai alla trentaseiesima ristampa e alle oltre 500.000 copie vendute in tredici lingue differenti.

I design pattern presentati nel libro seguono la stessa filosofia dei pattern architetturali di Alexander: coniugano vincoli a libero arbitrio, applicati però alla programmazione. Ad esempio, i pattern descritti dagli autori non sono presentati in un linguaggio di programmazione definito: infatti, viene usato uno pseudo linguaggio simile al Java o al C++ che permette a chiunque se ne intenda un minimo di programmazione di capire quale sia la funzione che ciascun elemento del pattern svolge, senza però costringere il programmatore a usare un determinato linguaggio se volesse usare il pattern nel proprio programma.

La pubblicazione del libro causò un'enorme ondata di entusiasmo e interesse per la "nuova" metodologia introdotta dalla "Gang of four": nell'anno seguente si tenne già la

prima conferenza dedicata esclusivamente ai design pattern informatici, seguita dopo pochi mesi dalla creazione del primo database online, il Portland Pattern Repository (il quale ha anche il primato di primo sito "wiki", cioè modificabile direttamente dagli utenti del sito). Questo raggruppamento di pattern, che spesso si riferiscono ad altri pattern esistenti, portò alla creazione del primo linguaggio di pattern informatico, che andò ad associarsi a quello esistente in campo architetturale.

Oggigiorno i pattern informatici esistenti sono tantissimi, e applicati agli ambiti più diversi dell'informatica: tuttavia, è possibile effettuare una classificazione tra di essi grazie alla differenza di astrazione presente. Il primo livello, sicuramente il più intuitivo, è quello dei pattern propriamente denominati di *disegno* (o, banalmente, *design pattern*): si riferiscono a problematiche riferite alla programmazione ad oggetti, e sono sostanzialmente quelli presenti nel libro della "Gang of four" e tutti i pattern creati in seguito che si mantengono staccati dal linguaggio di programmazione pur riferendosi a problemi strettamente legati a essa.

I pattern di più basso livello, riferiti specificamente a una tecnologia di implementazione o a un linguaggio ben determinato, sono chiamati *idiomi* e hanno il vantaggio evidente di poter sfruttare i punti di forza del contesto ben definito a cui sono rivolti; questo però va a scapito della riusabilità del pattern: è possibile che lo stesso pattern possa essere applicato anche in altri ambiti con altri linguaggi o tecnologie, ma è una circostanza puramente casuale in quanto il pattern stesso non è stato creato con questo scopo.

Il terzo livello è infine costituito da quei pattern chiamati *architetturali*, i quali hanno il massimo livello di astrazione: le problematiche a cui si riferiscono non sono minimamente collegate alla programmazione, ma si riferiscono, come suggerisce il nome, ad aspetti

riguardanti un sistema informatico nel suo complesso intrico di relazioni e dipendenze tra parti e sottosistemi componenti l'architettura di un progetto informatico.

Con questi tre livelli i design pattern praticamente hanno un utilizzo in qualunque momento e a qualunque livello dello sviluppo di un nuovo software, e la loro conoscenza, se non il loro utilizzo, è diventato imprescindibile dall'ingegneria del software o dalla programmazione di base.

2.1 – I Design Pattern e le ontologie

Il passo tra l'uso dei design pattern nell'ingegneria del software al loro uso nelle ontologie fu decisamente minore della ventina d'anni che furono necessari per passare dal libro di Christopher Alexander al libro della "Gang of Four". Bastarono pochi anni per i ricercatori e gli sviluppatori di ontologie si accorgessero delle potenzialità che i pattern offrivano anche nel loro campo, e difatti già alla fine degli anni '90 cominciarono ad apparire le prime pubblicazioni accademiche che collegavano il mondo delle ontologie a quello dei design pattern. L'enorme interesse suscitato nella comunità dello sviluppo di software da parte dei metodi di Alexander si propagò in fretta nel mondo delle ontologie. Nel 1999 già usciva una pubblicazione che già utilizzava la sigla che ancora oggi si usa per indicare i pattern ontologici, "Ontological Design Patterns for the Integration of Molecular Biological Information", mentre nei due anni successivi vennero pubblicati almeno due *paper* che

trattavano la possibilità di elaborare strutture semantiche in modo da poterle riutilizzare in altre situazioni simili a quella originale, definendole ovviamente design pattern.

Nel primo decennio del ventunesimo secolo quindi la produzione di articoli e la ricerca sui design pattern ontologici era ormai lanciata: pochi anni dopo, nel 2006, veniva già sovvenzionato dalla commissione europea il progetto NeOn, il quale aveva l'obiettivo di "[…] advance the state of the art in using ontologies for large-scale semantic applications in the distributed organizations. Particularly, we aim at improving the capability to handle multiple networked ontologies that exist in a particular context, are created collaboratively, and might be highly dynamic and constantly evolving" [1].

Il progetto aveva una durata di 4 anni, e si è concluso ormai un anno fa. Ha però prodotto risultati notevoli: oltre alle quasi trecento pubblicazioni edite nel corso del progetto, è stato rilasciato un toolkit basato sulla piattaforma di Eclipse, il quale è in grado di supportare lo sviluppo e il mantenimento di ontologie legate anche a domini molto diversi (grazie all'utilizzo di plugin in grado di differenziare le caratteristiche del toolkit), e la realizzazione di quattro siti atti a supportare in maniera indiretta l'utilizzo del tool. Questi tre progetti, "Ontology Metadata Vocabulary", "Cupboard ontology repository" e "Ontology Design Patterns repository" aiutano lo sviluppo e il mantenimento di ontologie in modi diversi: il primo sito è, come suggerisce il nome, un vocabolario scaricabile di metadati ontologici con lo scopo di favorire se non uno standard condiviso, almeno una nomenclatura condivisa su come indicare e definire i diversi elementi delle ontologie. Il secondo sito è, sempre deducibile dal nome, un motore di ricerca online, con funzioni

[1]: http://www.neon-project.org/nw/Welcome_to_the_NeOn_Project

semantiche, che permette di recuperare velocemente ontologie da un database online; il terzo e ultimo progetto è quello su cui si è incentrata la maggior parte del lavoro svolto per la realizzazione di questo elaborato.

Il progetto Ontology Design pattern è nato quindi come una parte del più grande progetto NeOn, e lo si può trovare all'indirizzo www.ontologydesignpatterns.org: il suo scopo consiste ancora oggi nel creare un repository online di ontologie aperto a chiunque voglia contribuire; ha inoltre l'obiettivo di fare in modo che, grazie all'attenta analisi e valutazione dei pattern proposti dalla comunità, si sviluppi una certa ufficialità legata ai pattern pubblicati nel catalogo ufficiale del portale. In sostanza, quindi, il gruppo che attualmente gestisce il sito punta non solo a creare un portale aperto alle proposte di chiunque voglia collaborare, ma anche a esser riconosciuto come organo ufficiale per la certificazione di design pattern ontologici efficienti e riutilizzabili. Il sito quindi, pur essendo accessibile a tutti, e tutti possono proporre nuovi pattern, pone quindi dei requisiti di qualità molto severi prima di effettivamente pubblicare nel catalogo ufficiale del portale qualunque pattern.

Il sito, che oltre ai pattern ontologici offre anche molto materiale didattico a riguardo, è gestito da un comitato editoriale composto da circa una ventina di persona, con a capo due editori, il professor Aldo Gangemi e la dottoressa Valentina Presutti. A questo primo comitato è affiancato un secondo, il comitato di qualità, specificamente creato per occuparsi della valutazione e della certificazione dei design pattern che vengono proposti dagli utenti del sito. Il portale è gestito in modo che dello stesso pattern possono esser pubblicate più versioni (per poter apportare modifiche alla proposta originale senza che

essa vada perduta), e per ognuna di esse è possibile visualizzare separatamente le recensioni che il proprio pattern ha ricevuto dai membri del comitato di qualità.

Il lavoro dei comitati comunque è appena all'inizio: a fronte dei più di 100 pattern proposti e disponibili per la consultazione sul sito, a oggi nessuno di essi è ancora stato ufficialmente inserito nel catalogo ufficiale del portale. Come vedremo, lo studio si sta ora concentrando su una tipologia di pattern, identificata dal comitato di qualità stesso: prima di passare alla creazione di specifici esempi, infatti, il comitato editoriale ha preferito elaborare diverse categorie di pattern per poter meglio svolgere il lavoro successivo concernente lo sviluppo di nuovi schemi ricorrenti. Vediamo quindi ora le categorie individuate e definite presenti a oggi sul sito.

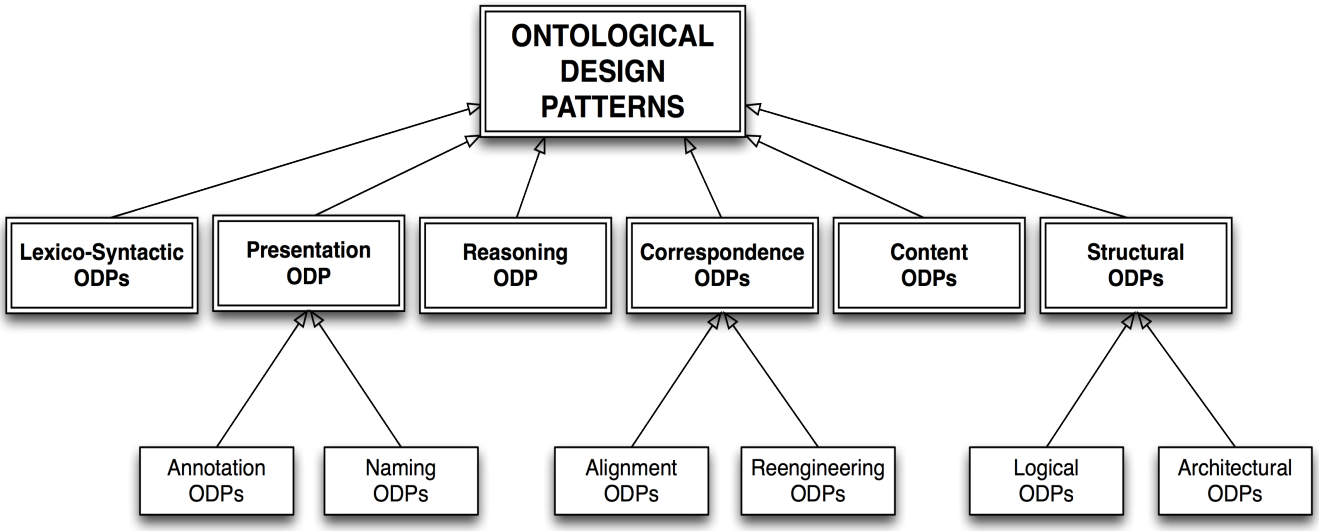
Capitolo 3 – Le categorie di design pattern ontologici

In questa sezione quindi analizzeremo il lavoro svolto finora dal gruppo di ricercatori sul già citato sito www.ontologydesignpatterns.org, dedicandoci principalmente alle varie categorie di pattern proposte finora e ai singoli pattern esistenti ad oggi. Queste categorie, identificate dal comitato di qualità, vogliono essere rappresentative in maniera esaustiva di tutte le possibili sfaccettature di ogni design pattern ontologico. A oggi i ricercatori hanno individuato 6 famiglie di pattern, comprendenti a loro volta alcune sottoclassi di pattern necessarie a rappresentare in maniera più specifica l'area a cui si riferiscono.

Una nota di lettura: d'ora in poi, per rapidità di lettura e per evitare noiose ripetizioni, i design pattern ontologici verranno indicati con la sigla ODP, formato dalle iniziali della rispettiva forma inglese del termine.

Le categorie a oggi sono:

- ODP Strutturali (divisi in Logici e Architetturali)
- ODP di Corrispondenza (divisi in Re-ingegnerizzazione e Allineamento)
- ODP di Contenuto
- ODP di Ragionamento
- ODP di Presentazione (divisi in Annotazione e Nominativi)
- ODP Lessico-Sintattici



Rappresentazione grafica delle categorie di ODP

Per scelta degli stessi comitati che gestiscono il sito, attualmente la ricerca è focalizzata sugli ODP di Contenuto, con già pianificato l'obiettivo di occuparsi in seguito degli ODP di Re-ingegnerizzazione e poi Logici. Questo ha portato ovviamente a una profonda diversità di materiale tra vari tipi di pattern: mentre per quelli di contenuto si conta ormai quasi un centinaio di proposte, per altri non solo non ne esiste neanche una, non esiste neanche la possibilità per gli utenti e i ricercatori di proporre degli ODP di quella categoria. Ciò ha ovviamente limitato le possibilità di approfondimento su alcune categorie, su cui è possibile soltanto fornire la descrizione della stessa fornita dal comitato editoriale del sito ma nessun esempio concreto che possa aiutare la comprensione della tipologia di design pattern che dovrebbero far parte della stessa.

Si è scelto quindi, anche a fronte di questa disparità di materiale, di affrontare le diverse categorie in ordine di complessità, analizzando per prime le meno documentate e quindi più veloci da descrivere, e tenendo per ultima gli ODP di Contenuto, su cui si sta concentrando la gran parte del lavoro dei comitati e dei singoli ricercatori. Su quest'ultima

categoria è stato anche possibile analizzare il metodo da seguire per proporne uno nuovo, in quanto è una procedura ben standardizzata, quasi ipotizzabile come un meta-pattern per la sottomissione di nuovi ODP.

3.1 – ODP di Presentazione

Questa categoria è stata creata con lo scopo di racchiudere tutti quei possibili design pattern volti a rendere più semplice la lettura di ontologie e la loro comprensibilità, ovviamente dal punto di vista dell'utente umano. Sostanzialmente, i pattern che ricadono in questa categoria sarebbero semplicemente delle buone pratiche di scrittura per ontologie, che, una volta valutate e approvate dal comitato, diventerebbero standard riconosciuti per la creazione di ontologie, con lo scopo di migliorarne la diffusione grazie a una maggiore comprensibilità.

La categoria è suddivisa a sua volta in pattern di Annotazione e pattern Nominativi. La prima sottocategoria comprende i pattern che, attraverso l'utilizzo di commenti e note, migliorano la comprensibilità dell'ontologia, ma soprattutto dei suoi elementi costitutivi. Un esempio generale, non ancora formalizzato in un pattern, potrebbe essere l'utilizzo delle etichette e dei commenti contenuti in RDF Schema. Purtroppo però per la categoria degli ODP di Presentazione non è ancora possibile presentare pattern, e quindi anche se esistono già delle tecnologie di utilizzo da cui si potrebbero sviluppare nuovi pattern, dal sito non è possibile ottenere nessun pattern ufficiale.

La seconda sottocategoria, i pattern Nominativi, hanno una funzione molto simile ai loro "fratelli", solo che focalizzano la loro attenzione non sui commenti e sulle annotazioni che

dovrebbero accompagnare un'ontologia con lo scopo di facilitarne la comprensione, ma piuttosto sui nomi dati alle ontologie e a i loro elementi costitutivi. Anche in questo caso c'è un esempio esistente, anche se non ancora formalizzato in un pattern: la costruzione richiesta per il namespace per le ontologie, in cui è raccomandato indicare un URI che punti all'organizzazione che pubblica l'ontologia in questione, seguito da un preciso riferimento a una directory online su cui trovare l'ontologia stessa; è comunque indicato come punto importante di eventuali pattern l'esistenza di una indicazione su come codificare la versione dell'ontologia, per mantenere una chiara indicazione dell'ontologia ed evitare ambiguità dovute a versioni meno aggiornate della stessa.

Come già detto, di questa categoria non esistono esempi proposti da ricercatori, ma soprattutto non esiste neanche la possibilità di proporne uno, per cui è stato impossibile anche analizzare un eventuale meta-pattern elaborato dai comitati per la proposta di nuovi ODP.

3.2 – ODP di Ragionamento

Questa seconda categoria di pattern è volta a raggruppare tutti gli ODP rappresentanti tutti quei processi logici atti a ottenere un preciso risultato di ragionamento automatico. Sono infatti delle applicazioni di ODP Logici (che vedremo più avanti) con lo scopo di ottenere dei precisi risultati, basati su un comportamento già implementato in diversi sistemi di ragionamento, per l'appunto. Gli esempi più famosi e conosciuti sono i processi logici di classificazione, sussunzione, ereditarietà, reificazione, per dirne alcuni: sono sostanzialmente dei procedimenti logici non riconducibili a un sistema logico o informatico

ben preciso. Ovviamente questa è la definizione generica della categoria, poi i futuri pattern che la popoleranno dovranno essere più precisi indicando a quale sistema di ragionamento automatico esistente fanno riferimento. Un utilizzo possibile di questi pattern è indicato dagli stessi autori del sito: indicando nelle prime parti dell'ontologia, ad esempio all'inizio del file OWL che la descrive, un tipo di ODP di Ragionamento, sarebbe possibile per un sistema automatico già sapere a quel punto quali procedimenti di ragionamento sono applicabili a quell'ontologia per poter eseguire query o altre operazioni logiche sulla stessa, aiutando così a velocizzare e ad automatizzare il processo.

Anche di questa categoria purtroppo non esistono né esempi né metodi per proporne una nuova, per cui i design pattern appartenenti alla categoria rimangono ancora allo stato ipotetico.

3.3 – ODP Lessico-Sintattici

In questa categoria di pattern ricadono tutti quegli ODP che rappresentano ontologicamente delle strutture sintattiche presenti nelle diverse lingue del linguaggio naturale. La loro utilità è legata principalmente alla lettura e all'interpretazione del significato da parte di sistemi automatici: l'esistenza di pattern di questo tipo permetterebbe a questi sistemi di poter estrarre ulteriori informazioni riguardo al significato delle frasi del linguaggio naturale che vengono analizzate. Attraverso questi pattern sarebbe quindi possibile dare una forma astratta a strutture sintattiche ricorrenti nel parlato, permettendo così di generalizzare e rendere riconoscibili delle forme idiomatiche altrimenti difficilmente riconoscibili e comprensibili. Principalmente per ora

però i pattern proposti sono utilizzati come supporto ad altri ODP Logici o di Contenuto, svolgendo la funzione di supporto per il riconoscimento di frasi del linguaggio naturale che descrivono o rappresentano casi d'uso degli ODP a cui si riferiscono. Questi ultimi però non sono ancora tutti presenti sul sito: alcuni sono solo stati proposti in un documento prodotto durante il progetto NeOn, ma si suppone che presto questi pattern verranno aggiunti a quelli proposti e in attesa di valutazione.

Di questa categoria è già possibile presentare pattern da sottoporre al giudizio del comitato di qualità e alle revisioni degli utenti: daremo prima quindi una breve descrizione del procedimento da seguire per proporre un nuovo pattern.

Si tratta di un semplice form online da compilare, con campi opzionali e campi necessari (struttura per le nuove proposte utilizzata poi per tutti i tipi di pattern) che possono variare in numero e tipo, con però una restrizione su ciò che ciascuna casella di testo può contenere: si va dalle più banali string (cioè si può inserire qualunque tipo di dato o testo) ai casi in cui è obbligatorio inserire un URI fino ai casi in cui non si è liberi di inserire ma solo di scegliere tra alcune opzioni predefinite. La maggior parte delle caselle di testo è comunque gestita come una serie di proprietà come se fossero delle componenti di un unico grande RDF Schema: ogni casella di testo che viene compilata rappresenta quindi una proprietà del pattern che si sta creando, permettendo così di poter gestire i pattern creati, le loro modifiche e il tracciamento della versione in maniera rapida ed efficace. Ad ogni pattern è quasi sempre richiesto di associare una o più immagini che facilitino la comprensione del pattern, siano esse immagini esemplificative di casi d'uso dell'ODP o rappresentative della visione più generale del pattern stesso: nel caso degli ODP Lessico-

Sintattici l'immagine è opzionale, mentre, come vedremo, nel caso degli ODP di Corrispondenza sono necessarie più di una immagine come allegati a ogni singolo pattern.

Torniamo ora agli ODP Lessico-Sintattici: in questo caso all'utente è richiesto di compilare obbligatoriamente le caselle riguardanti il nome del pattern (per ovvi motivi), il linguaggio a cui il pattern fa riferimento (le forme idiomatiche dipendono ovviamente dalla lingua usata; sul sito esistono alcuni ODP rappresentanti la stessa forma in inglese e in spagnolo), l'intento del pattern (in cui bisogna descrivere, in maniera approfondita, l'obiettivo che si vuole raggiungere con la creazione dell'ODP in questione) e la corrispondenza con cui si identifica questo pattern (cioè se questo singolo ODP Lessico-Sintattico si riferisce a uno, due o più altri ODP; in questo caso la scelta è limitata a queste tre diverse opzioni). Inoltre, per questi tipi di pattern è caldamente consigliato l'invio di almeno un caso d'uso che possa esemplificare in maniera concreta la struttura linguistica a cui ci si sta riferendo: non solo esempi in linguaggio naturale, ma anche una formalizzazione del formato generale della forma idiomatica, codificata tramite una serie di elementi molto simili a quelli usati nelle espressioni regolari. Riportiamo alla pagina successiva la tabella presente sul sito comprendente queste simbologie e abbreviazioni.

Vediamo ora quindi degli esempi tratti dalle proposte fatte finora per ODP di questa categoria. Analizziamo quindi come primo esempio "*Lexico Syntactic ODPs corresponding to Participation ODP* ": come indica il nome, questo pattern è stato proposto come supporto al pattern di Contenuto chiamato Participation, il quale rappresenta la partecipazione di un oggetto (inteso nel senso ontologico del termine) a un determinato evento. L'ODP Lessico-Sintattico ha quindi lo scopo di riconoscere le forme idiomatiche inglesi che stanno a indicare questo tipo di interazione tra entità ed eventi. Per chiarire

meglio, nello stesso pattern vengono forniti i seguenti esempi: “Players are involved in competitions” e “Engineering project managers participate in writing specifications, researching, and selecting suppliers and materials”. Come si può notare, entrambe le costruzioni presentano un soggetto (corredato eventualmente di aggettivi), seguito da una componente verbale che indica partecipazione e terminano con una o più parole che si riferiscono a un qualche tipo di evento.

Questi esempi pratici sono anche corredati dalla formalizzazione mostrata nella tabella seguente: nel caso dell’ODP in considerazione, la “formula” rappresentante questa forma idiomatica fornita è

NP<object> 20 ras20rce20i/take part in/be involved in [(NP<event>,) and] NP<event>*

Come si può notare, in questo tipo di formalizzazione sono presenti elementi facilmente riconducibili (e quindi interpretabili) ai simboli delle espressioni regolari e alle categorie sintattiche usate nell’elaborazione del linguaggio naturale. La “formula” ci dice in sostanza che se la frase è composta da una parte nominale (che può comprendere il soggetto della frase più eventuali aggettivi) più una parte verbale (nella formula vengono indicati solo i più comuni verbi inglesi che indicano partecipazione; l’elenco sicuramente non è completo e si potrebbe espandere) e un costrutto indicante una lista non nulla di elementi (un costrutto molto comune nelle espressioni regolari) separati dalla virgola e dalla congiunzione “and” per l’ultimo termine dell’elenco.

I pattern di questa categoria sono tutti molto simili: altri tipi possono essere l’ODP per la relazione di equivalenza tra classi (il caso portato ad esempio è “Poison dart frogs are also known as poison-arrow frogs”; una parte nominale seguita da una parte verbale che eguaglia il soggetto a una seconda classe) o quello per identificare la forma inglese

Simboli e Abbreviazioni	Descrizione
AP<…>	Adjectival Phrase. It is defined as a phrase whose head is an adjective accompanied optionally by adverbs or other complements as prepositional phrases. AP is followed by the semantic role played by the concept it represents in the conceptual relation (for instance, property) in angle brackets
CATV	Verbs of Classification. Set of verbs of classification plus the preposition that normally follows them. Some of the most representative verbs in this group are: classify in/into, categorize in/into, subcategorize in/into, group in/into, fall into.
CD	Cardinal Number
CN	Class Name. Generic names for semantic roles usually accompanied by preposition. Two main groups have been identified: CN conveying classification (CN-CATV) (class, group, type, subtype, subclass, category, species, family, order, example) and CN conveying mereological relations (CN-PART) (part, set, member, constituent, component, element, piece, item, layer). If not otherwise specified, CN can include generic names such as period, area or phase.
PART	Verbs of Mereology. Set of verbs conveying the relation existing between a whole and its parts. Some of the most representative ones are: contain, form part of, consist of, comprise, be composed of, be made up of, be formed of, be part of, be constituted of
NP<…>	Noun Phrase. It is defined as a phrase whose head is a noun or a pronoun, optionally accompanied by a set of modifiers, and that functions as the subject or object of a verb. NP is followed by the semantic role played by the concept it represents in the conceptual relation in question in <…>, e.g., class, subclass, part, property, value, object, etc.
PARA	Paralinguistic symbols like colon, or more complex structures such as as follows, etc., that introduce a list
PREP	Prepositions
QUAN	Quantifiers such as all, some, most, many, several, every, etc.
REPRO	Relative pronouns such as that, which, whose
()	Parentheses group two or more elements
*****	Asterisk indicates repetition
[]	Elements in brackets are meant to be optional, which means that they can be present either at that stage of the sentence or not. By default of appearance, the semantic of the pattern remains unmodified
NEG	Elements preceded by this abbreviation should not appear in the pattern

Tabella riassuntiva delle abbreviazioni e dei simboli usati negli ODP Lessico-Sintattici

che esprime ereditarietà da più di una classe (l'esempio è "Amphibians are water-living and land-living animals"; una parte nominale indicante una classe collegata da una verbale a due super-classi non identiche). In ogni caso, esclusi i due pattern presenti riferiti alla lingua spagnola, ogni esempio è corredato di un caso e di una formalizzazione creata seguendo le regole della tabella alla pagina precedente. L'unica altra nota rilevante è che non tutti i pattern proposti, come dicevo, sono collegati a un altro ODP presente sul sito: gli ultimi due esempi portati infatti si riferiscono a una serie di modelli teorizzati in un documento prodotto dal progetto NeOn, in cui erano contenuti una serie di modelli teorici di possibili ODP ma che non sono ancora stati tutti trasformati in proposte di ODP da valutare.

3.4 – ODP di Corrispondenza

La prossima categoria di ODP è rivolta invece solo al mondo delle ontologie: i pattern che ricadono in questo gruppo servono esclusivamente alla gestione e alla creazione di nuove ontologie. Questa categoria infatti si divide in due sottogruppi: gli ODP di Allineamento e quelli di Re-ingegnerizzazione. Vediamo ora le due categorie più in dettaglio.

ODP di Allineamento

Lo scopo dei pattern raccolti in questa categoria è quello di fungere da "collegamento" tra diverse ontologie: in pratica, tramite una serie di entità ontologiche, permettono di stabilire equivalenze tra due ontologie con ruoli e proprietà diverse, così da evidenziare punti in comune e permettere, appunto, l'allineamento tra ontologie differenti.

L'esempio esemplificativo di questo tipo di confronto fra ontologie pubblicato sul sito è riportato nell'immagine seguente: date due ontologie diverse, ma che si riferiscono allo stesso oggetto del mondo reale (in questo caso un vino di Bordeaux), tramite le due entità evidenziate in verde è possibile mettere in relazione le entità delle due diverse

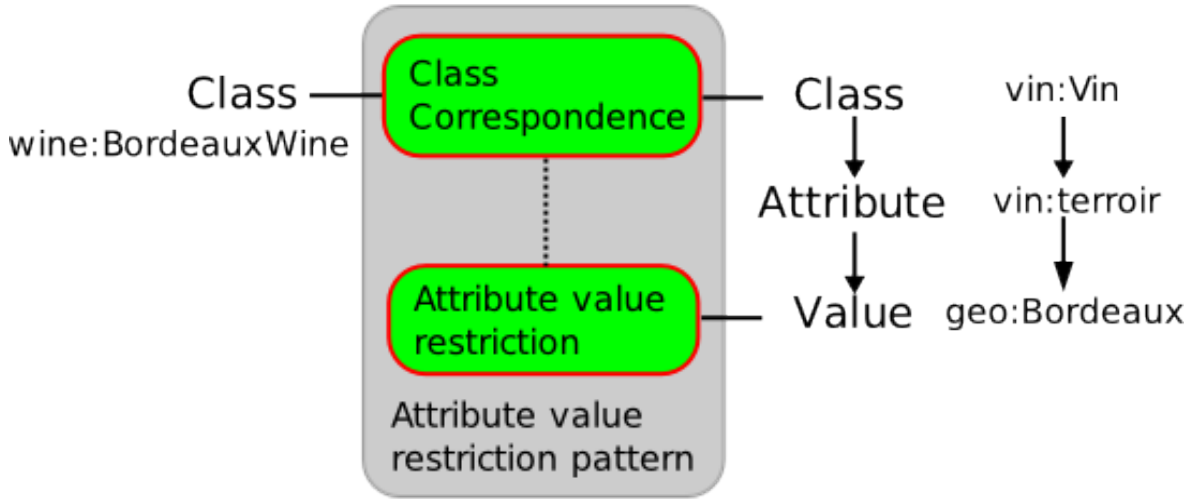


Figura esemplificativa del ruolo di un ODP di Allineamento

ontologie. Quindi la classe BordeauxWine della prima ontologia ora è messa in relazione con le entità della classe Vin con attributo terroir posto al valore "Bordeaux" della seconda ontologia.

Un esempio concreto estratto da quelli presenti sul sito è simile all'esempio precedente: un pattern che permette l'allineamento tra una classe in una prima ontologia e un'intersezione di più classi in una seconda ontologia. Il caso riportato sul sito descrive una classe CanadianCitizenByBirth equiparata all'intersezione delle classi PersonBornInCanada e PersonWithCanadianParents. Gli altri ODP finora proposti son tutti sullo stesso genere: equivalenza tra una classe e un'intersezione di classi, tra una classe e un'altra ma con

restrizione sul tipo di attributo o sul valore dello stesso; comunque ogni esempio di questi pattern ha un'utilità possibile solo rivolta al mondo delle ontologie.

Anche per questa categoria si ha il solito schema di proposta: una form da compilare, con campi più o meno obbligatori. In questo caso però non è prevista nessuna formalizzazione del concetto, tutte le parti possono esser compilate in linguaggio naturale senza bisogno di parti in codice o altro. Viene solo caldamente consigliata la presenza di un'immagine allegata al pattern (molto utile per facilitarne la comprensione), anche se i dieci pattern finora proposti comprendono tutti un esempio di soluzione scritta in XML/RDF. Va considerato anche il fatto che per ora gli studi si stanno concentrando sui pattern di Contenuto: in un futuro, quando la ricerca si sposterà su questa categoria (e le altre con requisiti così poco definiti), probabilmente verranno stabiliti criteri e standard da rispettare anche per questi pattern.

ODP di Reingegnerizzazione

Questa sottocategoria dei pattern di Corrispondenza è volta ad includere quegli ODP che permettono di generare nuove ontologie: lo scopo dichiarato infatti è quello di raggruppare tutti quei pattern che, partendo da un modello di base non codificato secondo regole ontologiche, permettano di generare un'ontologia corrispondente al concetto espresso dal modello non ontologico. Ad esempio, i modelli di partenza possono essere un modello UML, una struttura linguistica, un qualsiasi modello di classificazione. Questo però rende la categoria una delle più complesse da analizzare.

Per proporre un nuovo pattern di questo tipo, infatti, è suggerito caldamente di inserire una serie di immagini atte a facilitare la comprensione del meccanismo di trasformazione

che il pattern applicherebbe. Per rendere meglio l'idea, il gruppo di studio del sito obbliga, se si vuole sottoporre un pattern per il giudizio, ad allegare ben 6 immagini: una per la rappresentazione grafica del modello non ontologico di partenza, una per la rappresentazione dell'ontologia generata, una per un diagramma rappresentante i passi che il pattern percorre durante la trasformazione, più una copia di ciascuna immagine applicata a un esempio concreto. Inoltre, nella consueta form da compilare per la sottomissione di un nuovo ODP, uno dei campi obbligatori richiesti è un riferimento web a una pagina che contenga il modello di partenza a cui si vuole applicare la trasformazione.

Vediamo quindi ora un esempio di pattern tratto dai dodici presenti sul sito: verrà preso in considerazione il pattern "*Pattern for re-engineering a term-based thesaurus, which follows the record-based data model, into an ontology schema*"; in sostanza il pattern si propone di estrarre da un thesaurus, cioè un dizionario di sinonimi, una rappresentazione ontologica delle relazioni esistenti tra le parole.

Lo schema di un thesaurus tipo considerato per il pattern è il seguente:

Termine	**BT**	**NT**	**RT**	**UF**
Term1	BTTerm1	NTTerm1 NTTerm2	Term2	UFTerm1
Term2	BTTerm2	NTTerm3 NTTerm4 NTTerm5 NTTerm6 NTTerm7	RTTerm3 RTTerm4 RTTerm5	

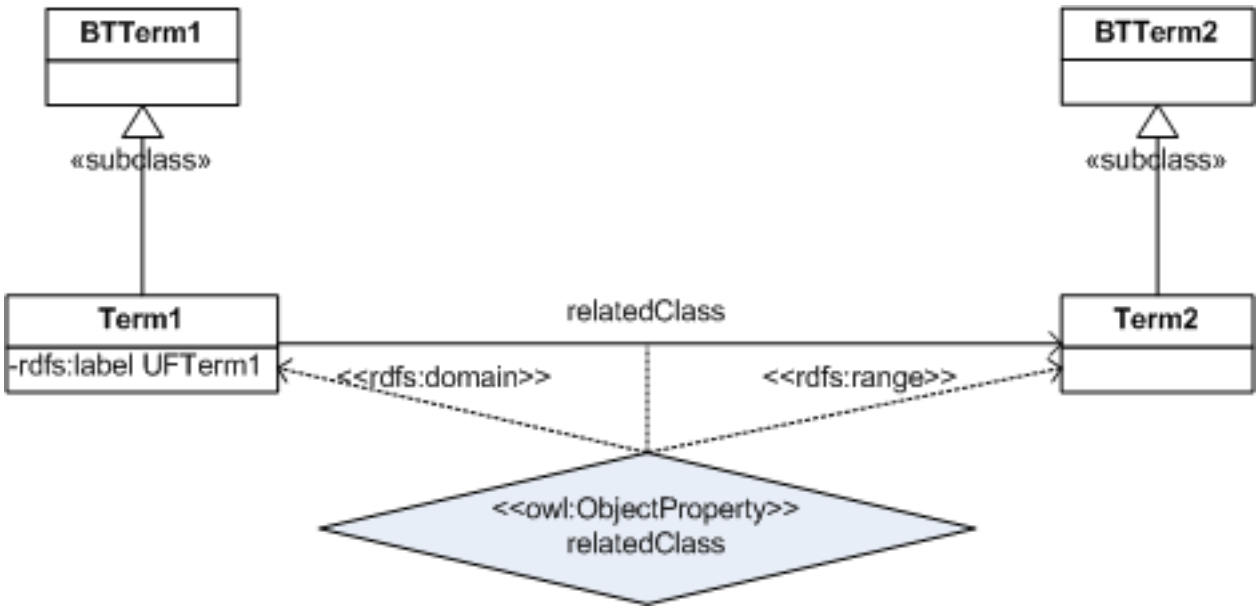
La sigla BT indica il Broader Term, cioè la parola che descrive in maniera più ampia il campo a cui il termine si riferisce; NT sta per Narrow Term, e serve a indicare parole quasi sinonime dell'originale ma che evidenziano un aspetto diverso; RT sta per Related Term, e

sono i veri e propri sinonimi della parola cercata; UF sta per Used For e indica l'area di significato a cui solitamente si riferisce la parola cercata. Per meglio chiarire, è riproposto l'esempio concreto di questa tabella come pubblicato sulla pagina di questo ODP, con la parola "Competence" come punto di partenza della ricerca di sinonimi.

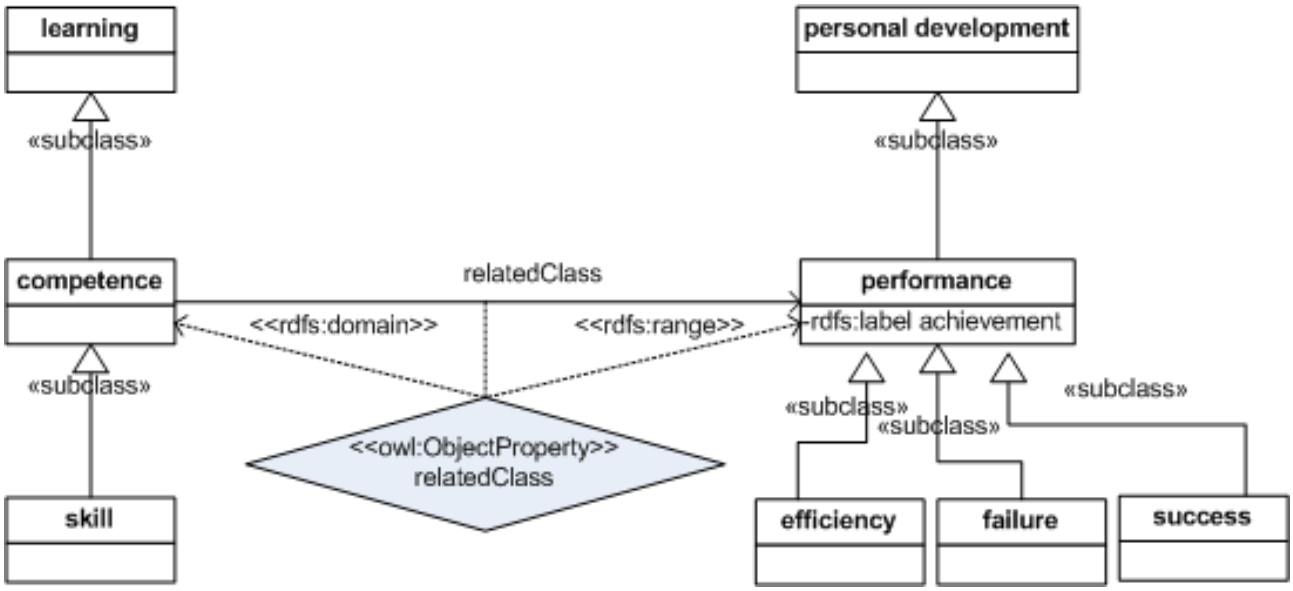
Termine	**BT**	**NT**	**RT**	**UF**
Competence	Learning	Skill	Aptitude Know how Knowledge Performance	
Performance	Personal development	Efficiency Failure Success	Competence Productivity	achievement

Questa struttura, così come l'esempio, sono tratte dall'European Training Thesaurus, recuperabile all'indirizzo http://libserver.cedefop.europa.eu/ett/en/.

Quindi, il pattern in questione si propone di costruire un'ontologia a partire da queste tabelle, per mettere in evidenza le relazioni semantiche tra le diverse parole. L'ontologia avrebbe dunque un aspetto simile:



Come si può vedere, le parole vengono definite come sottoclassi del loro Broader Term, e sussiste una relazione tra Term1 e Term2 in quanto sono elencate come Related Term ciascuna per l'altra. Come per la tabella precedente, ecco l'ontologia generata a partire dal semplice esempio di "competence" e "performance":

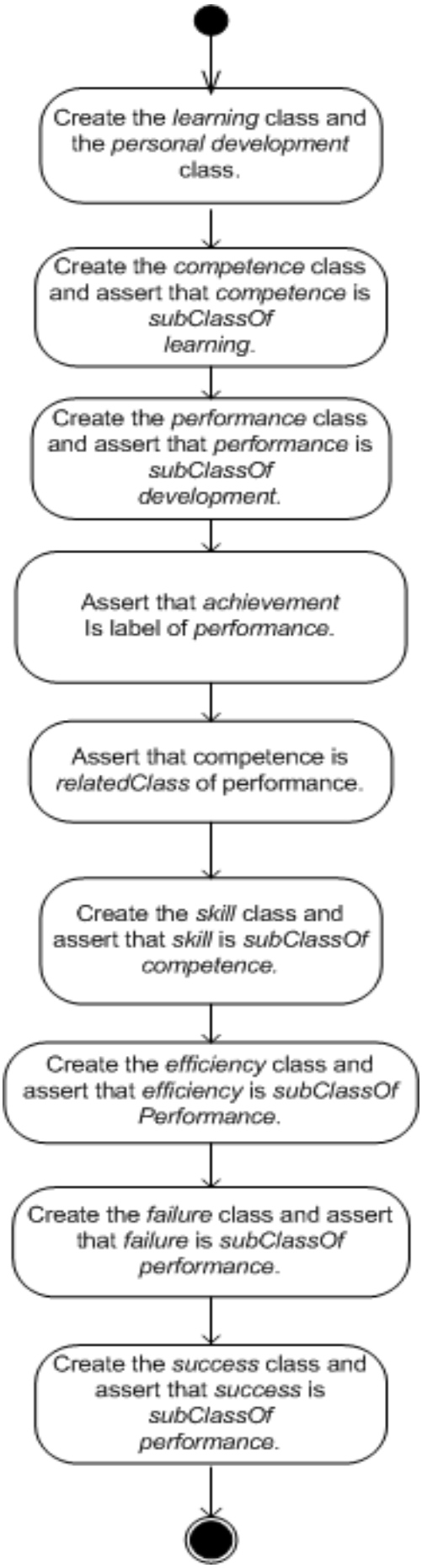


A conferma di quanto detto, si può notare la relazione esistente tra i due termini (sono elencati rispettivamente nella colonna RT di ciascuno), mentre i BT sono elencati come superclassi mentre i NT come sottoclassi. I termini UF si è scelto di indicarli come proprietà RDFS:Label, ed è ovvio specificare che esistono altrettante relazioni uscenti da "competence" e "performance" per quanti sono gli altri RT elencati in tabella, ma che per chiarezza del grafico non sono stati riportati.

Riportiamo, per completezza dell'esempio, anche l'elenco dei passi da seguire per trasformare la tabella in un'ontologia, e il grafo di flusso rappresentante il procedimento eseguito nella creazione dell'ontologia d'esempio basata sulle parole "competence" e "performance".

1. Identify the records which contain thesaurus terms without a broader term.
2. For each one of the above identified thesaurus terms Ti:
2.1. Create the corresponding ontology class, Ci class, if it is not created yet.
2.2. Identify the thesaurus terms, Tj , which are narrower terms of Ti. They are referenced in the same record which contains Ti.
2.3. For each one of the above identified thesaurus term Tj :
2.3.1. Create the corresponding ontology class, Cj class, if it is not created yet.
2.3.2. Using the external resource identify the semantics of the relation between Cj and Ci, and set up the relation identified.
2.3.3. Repeat from step 2.2 for Cj as a new Ci
2.4. Identify the thesaurus terms, Tr, which are related terms of Ti. They are referenced in the same record which contains Ti.
2.5. For each one of the above identified thesaurus terms Tr:
2.5.1. Create the corresponding ontology class, Cr class, if it is not created yet.
2.5.2. Using the external resource identify the semantics of the relation between Cr and Ci, and set up the relation identified.
2.5.3. Repeat from step 2.4 for Cr as a new Ci
2.6. Identify the thesaurus terms, Tq, which are equivalent terms of Ti. They are referenced in the same record which contains Ti.
2.7. For each one of the above identified thesaurus terms Tq:
2.7.1. Use the logical pattern proposed by Corcho et al.
2.7.2. Repeat from step 2.6 for Cq as a new Ci.

Algoritmo per l'applicazione dell'ODP di Reingegnerizzazione d'esempio



Grafo di flusso rappresentante la creazione dell'ontologia d'esempio

3.5 – ODP Strutturali

Questa categoria, una delle più fornite di esempi, riguarda due tipi di ODP: quelli riguardanti l'architettura delle ontologie, detti ODP Architetturali, e quelli puramente legati all'espressività logica di un linguaggio, detti per l'appunto ODP Logici.

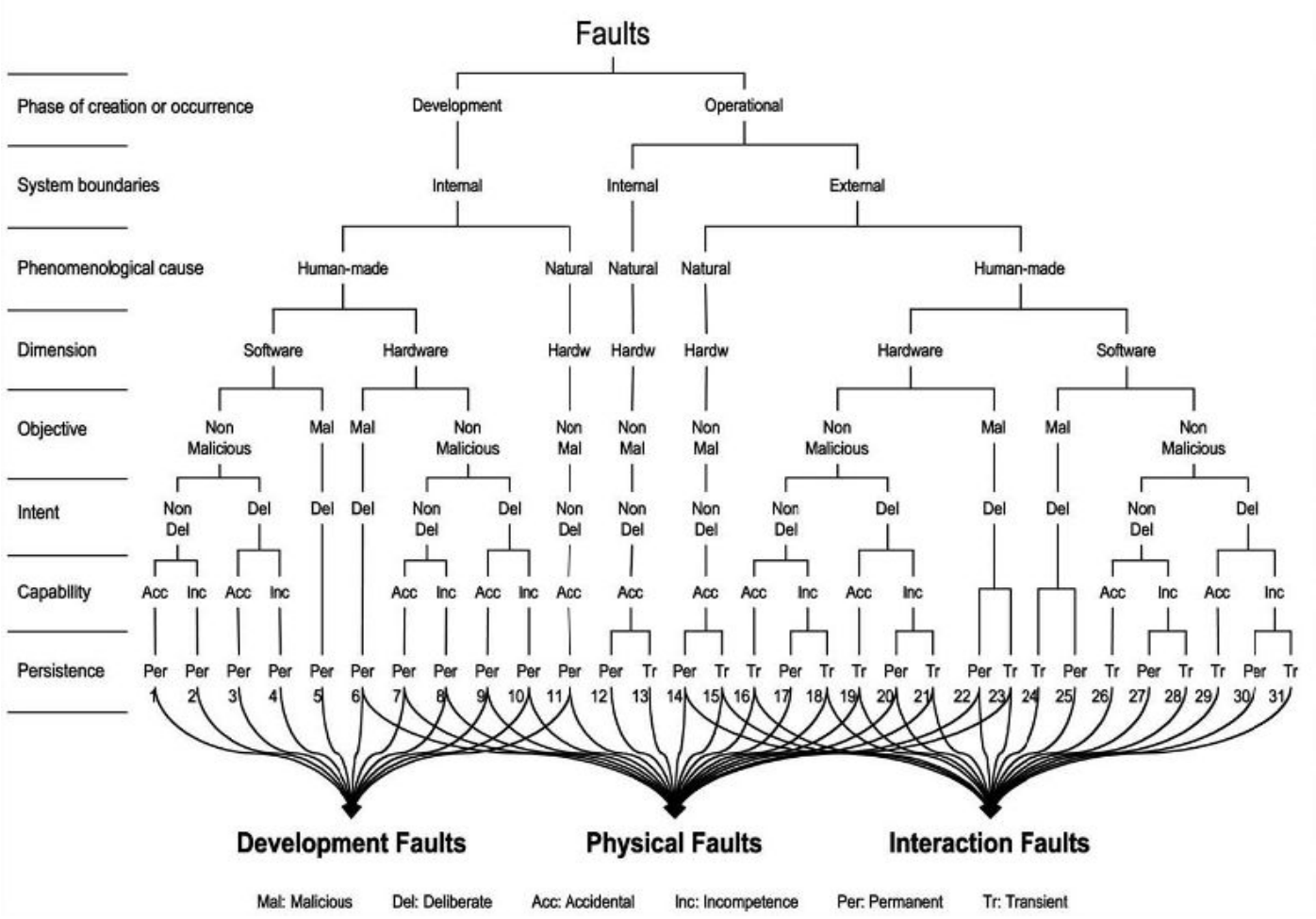
ODP Architetturali

Lo scopo dichiarato di questa categoria è quello di raggruppare tutti quei pattern che hanno l'obiettivo di definire e standardizzare "come una ontologia dovrebbe apparire". L'idea di questa categoria, e i suoi pattern, sono nati come scelte di design motivate da bisogni specifici, ad esempio la necessità di rientrare in certi limiti di complessità computazionale. Sono particolarmente utili come documenti di riferimento per i profani che si volessero avvicinare al mondo delle ontologie.

Come per le altre categorie finora analizzate, sia gli esempi sia la presenza di meta-pattern individuabili per la proposta di nuovi ODP è praticamente nulla: sul sito per ora esiste un solo pattern proposto per questa categoria, mentre i dati necessari da fornire assieme al pattern sono semplicemente nome, autore e mittente. Vediamo quindi l'unico esempio di pattern disponibile, che comunque è ben documentato e spiegato nonostante l'assenza di un processo di pubblicazione non standardizzato e controllato.

L'ODP Architetturale in questione è intitolato "*View Inheritance*", e si pone l'obiettivo di essere uno standard per la creazione di ontologie in cui si hanno dei concetti difficili da rappresentare a causa della complessità della loro definizione e della presenza di diversi

criteri alternativi di classificare queste astrazioni. Per meglio esprimere questo tipo di problema, l'autore del pattern ha deciso di portare come esempio una possibile classificazione degli errori, dei "fault", che posso capitare a un sistema informatico nell'arco della sua vita attiva: questi sono poi stati suddivisi per categorie, basandosi su diversi aspetti di volta in volta, creando così un albero di classificazione di ogni tipo d'errore possibile. Ecco quindi l'albero generato da questo procedimento: a sinistra possiamo vedere i criteri con cui è stata creata ogni generazione di rami dell'albero (da errori in creazione fino a errori umani nell'utilizzo), mentre i tre nodi sotto l'albero che raggruppano le foglie sono i concetti astratti a cui si riferivano i creatori del pattern.

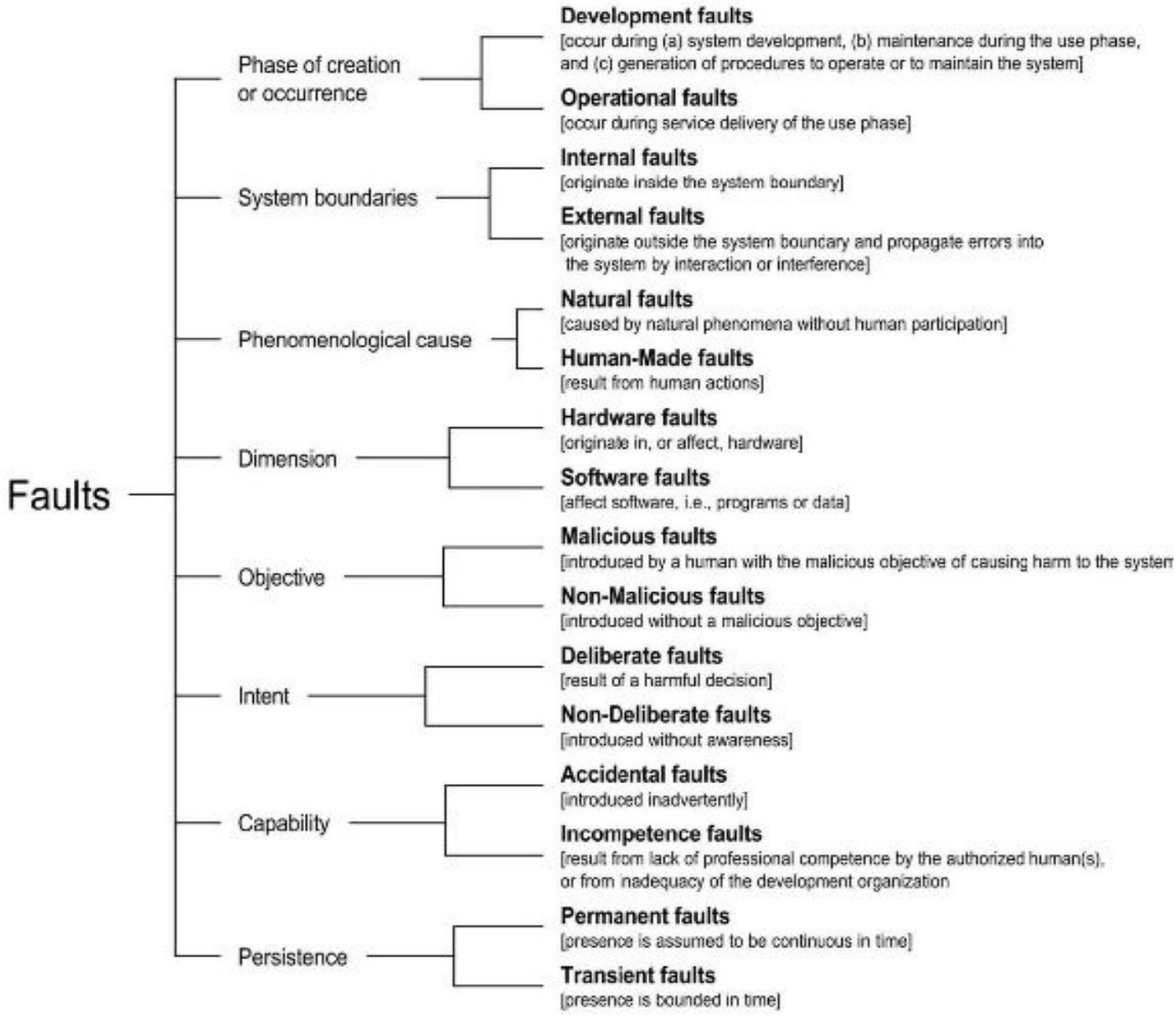


Albero degli errori di un sistema informatico

Questa classificazione sarebbe difficile se non impossibile da esprimere tramite il classico sistema della sottoclassi e dell'ereditarietà: come si può vedere dall'albero, ognuno dei tre concetti si sovrappone con gli altri,e soprattutto si riferisce a foglie che, seguendo il percorso dalla radice, sono tra di loro separate, se non addirittura appartenenti ad insiemi disgiunti. Inoltre, la possibilità di considerare una diversa prospettiva nell'approcciare il concetto di "system fault" potrebbe portare a un albero diverso ma non per questo errato, come dimostra l'albero alla pagina seguente, sempre fornito dall'autore del pattern in esame.

Questo problema di diverse prospettive viene quindi risolto da questo ODP con la creazione di tre apposite classi (dove con *i, j, x, y* ci si riferisce alla sequenza di numeri naturali):

- *Criterion_i*: queste classi sono volte a rappresentare ciascuno dei criteri alternativi di astrazione del concetto radice dell'albero, e rappresentano in sostanza i figli di primo livello della radice nell'ontologia risultante. L'elenco di questi criteri non è necessariamente esauriente o formato da coppie disgiunte di elementi.
- *Ci_Class_x*: queste classi servono a raffinare i concetti iniziali (quelli di primo livello), e rappresentano in sostanza gli aspetti presi in considerazione in quella prospettiva del concetto radice. Anche in questo caso, la lista può non essere esaustiva o disgiunta.
- *CiClass_xCjClass_y, Ci_Class_xClass_y*: queste classi sono il nocciolo dell'ODP, e rappresentano le categorie a ereditarietà multipla provenienti dalle diverse astrazioni del concetto radice.



Albero alternativo per la classificazione degli errori

L'applicazione di questo ODP porta quindi alla creazione di una ontologia come quella rappresentata nell'immagine alla pagina seguente. Come si può vedere, dal concetto radice "Faults" sono state esplose prima le diverse prospettive (corrispondenti alle classi *Criterion_i*), tra le quali possiamo notare la classe

Ontologia derivata dall'astrazione secondo diversi criteri del concetto di "Fault"

“MajorGroupFaults” derivante dal primo albero proposto, e la classe “BasicViewPoinFaults” derivante dal secondo albero portato ad esempio.

Si può notare come poi ciascuna prospettiva sia stata poi successivamente raffinata poi con le classi corrispondenti alle classi *Ci_Class_x*, fino a giungere al massimo della raffinazione con le foglie dell’albero ontologico. Le due classi indicate con “FaultType1” e “FaultType2” sono gli esempi delle classi *CiClass_xCjClass_y* e *Ci_Class_xClass_y*, che combinano diversi aspetti provenienti da diversi criteri di rappresentazione dello stesso concetto di base. Un esempio di queste ultime classi potrebbe essere la classe *ErroriDistrazione*: conterrebbe quindi errori provenienti come alcuni provenienti dalla prima astrazione dalla classe “IncompetenceFault” o altri provenienti dalla seconda dalla classe “InteractionFault”, e senza l’ausilio di questo ODP rappresentare queste classi con ereditarietà multipla sarebbe un’impresa difficile che porterebbe sicuramente a un numero di soluzioni diverse e non standardizzate.

ODP Logici

L’altra categoria in cui si dividono gli ODP Strutturali è quella dei pattern denominati Logici: si tratta di un gruppo in cui ricadono tutti quegli ODP il cui obiettivo è quello di rappresentare un costrutto logico indipendente dal contesto. Il loro ruolo è quindi quello di aiutare a risolvere problemi di costruzione di ontologie derivanti dal mancato supporto di una determinata struttura logica da parte del linguaggio o mezzo di espressione logica che si è scelto. Vedremo però questo tipo di pattern nel prossimo capitolo, interamente dedicato alla loro analisi.

3.6 – ODP di Contenuto

L'ultima categoria rimasta da analizzare è quella attualmente allo studio dei comitati editoriale e di qualità, e lo si può ben notare dal numero di pattern finora proposti dalla comunità di utenti, ben 88.

Questa categoria, in breve, racchiude i pattern rivolti alla soluzione di problemi concettuali, piuttosto che logici: sono definiti anche come istanze di pattern Logici. In sostanza, gli ODP di Contenuto sono strettamente legati al dominio, e quindi all'ontologia a cui si riferiscono: da ciò consegue che il vocabolario usato per crearli è esplicitamente non logico, e, anche se dovrebbero in teoria essere indipendenti da un linguaggio ontologico preciso, sono di fatto implementati in OWL per permetterne l'utilizzo e il ri-utilizzo come pattern. Si è in pratica rinunciato a un po' di portabilità degli ODP per avere la possibilità di un utilizzo effettivo e pratico degli stessi.

Vediamo quindi ora quali sono i requisiti di un ODP di Contenuto. Una prima novità rispetto alle altre categorie è la richiesta della presenza di un dominio: bisogna selezionarne uno tra gli esistenti, o crearne uno nuovo; una volta compiuta questa operazione, bisogna scaricare il file cpannotationschema.owl (si veda allegato A) e importarlo nel proprio file OWL contenente il pattern che si vuole proporre. E' infatti necessario e obbligatorio inviare il proprio pattern in questo formato, se si vuole che venga valutato ed eventualmente certificato dal comitato di qualità: come si è detto sopra, si è scelto di rinunciare a un minimo di portabilità per aver la possibilità di usare questi pattern come "reusable building blocks", definizione usata sul sito stesso per gli ODP di Contenuto.

Il file OWL, una volta modificato con l'import del file richiesto dal sito, deve comunque essere pubblicato online (necessariamente uno diverso da quello in questione) e l'indirizzo di dove lo si è pubblicato è uno dei dati richiesti per la proposta del pattern. E' inoltre e come sempre suggerito il caricamento di un'immagine che possa meglio esporre lo scopo del pattern, ma il fatto più notevole in questa categoria è che è possibile, compilando la form disponibile sul sito, generare direttamente un file OWL, compensando in questo modo la diminuzione di portabilità dei pattern proposti.

Il forte collegamento di questi pattern con il loro contesto è mostrato soprattutto dal numero e dalla varietà di domini finora pubblicati sul sito: per 88 esempi di ODP esiston già ben 41 domini, i quali spaziano dai prevedibili *Knowledge Engineering* o *Web 2.0* ai più "fuori tema" *Fishery* o *Agriculture*. Analizziamo alcuni di questi pattern proposti.

Partiamo da un esempio semplice e piuttosto legato al nostro contesto: il pattern "ActingFor", proposto dallo stesso professor Gangemi. Questo ODP ha il semplice scopo di rappresentare in OWL la relazione esistente tra un agente (sia esso una persona singola o una delegazione, o un'entità non fisica) e ciò che esso rappresenta, però limitato a un'entità sociale (cioè un'organizzazione o un qualche tipo di società). Ciò viene realizzato tramite due semplici classi OWL (*Agent* e *SocialAgent*: la prima può rappresentare un'entità fisica, la seconda no) e da due proprietà simmetriche *ActsFor* e *ActsThrough*, il cui nome ben denota il ruolo che svolgono nel pattern. Il file OWL, come appare sul sito, rappresentante il pattern è riprodotto nell'allegato B.

Un altro esempio, molto diretto e facilmente comprensibile, è il pattern "LinnaeanTaxonomy", sempre proposto dal professor Gangemi, che partendo dalla classificazione ad albero proposta da Linneo per le specie animali costruisce uno schema in

OWL per rappresentarla ontologicamente tramite otto classi, una per livello di tassonomia (una per specie, per classe, per famiglia etc) e quattro proprietà che rappresentano la relazione, diretta o meno, tra una classe e le sue super- o sotto-classi.

Si vede quindi come le classi proposte da ciascuno di questi pattern siano effettivamente molto legate al dominio a cui si riferiscono: difficilmente, infatti, si potrebbero riutilizzare le classi create in “LinnaeanTaxonomy” per un’altra ontologia senza un lavoro, anche se minimo, di modifica dei nomi attribuite alle entità ontologiche.

Si può notare come sia un esempio di come gli ODP di Contenuto siano istanze di ODP Logici: il corrispondente ODP logico di “LinnaeanTaxonomy” sarebbe un pattern astratto contenente le istruzioni per sviluppare una classificazione ad albero, dove ogni sotto-classe verrebbe generata dalla corrispondente super-classe per specializzazione.

Capitolo 4 – ODP Logici

Delle categorie qui finora esaminate, si è scelto di prendere in maggior considerazione ed approfondire quella riguardante gli ODP Logici. Si è scelta questa categoria perché è la più vicina all'ambito strettamente teorico riguardante lo sviluppo delle ontologie: gli ODP di Contenuto, anche se sono l'obiettivo dello studio attuale dei ricercatori, sono spesso troppo legati a un dominio specifico per poter essere considerati degli esempi sufficientemente validi; infatti, come già evidenziato in precedenza, è stato reso obbligatorio per la presentazione di un ODP di Contenuto l'implementazione OWL per poterlo rendere riusabile e nel contempo contenere l'eccessivo adattamento del pattern al dominio di riferimento. Inoltre, e soprattutto, i pattern di Contenuto sono definiti anche come istanze di pattern Logici applicati a un dominio in particolare: la scelta più ovvia, visto anche il lavoro maggiormente orientato a un approccio teorico fin qui svolto, è stata quella quindi di analizzare i pattern Logici e non quelli di Contenuto per poter mantenere l'alto livello di astrazione comune a tutta la ricerca svolta finora.

Il ruolo di questi pattern, come già accennato nel capitolo precedente, è quello di aiutare a risolvere problemi di costruzione di ontologie derivanti dal mancato supporto di una determinata struttura logica da parte del linguaggio o mezzo di espressione logica che si è scelto. Si tratta quindi di pattern di livello piuttosto alto, completamente scollegati da un contesto di utilizzo, e, quindi, concettualmente più vicini al mondo dell'ingegneria del

software. Ad esempio, se il linguaggio di rappresentazione scelto è OWL, e si ha la necessità di rappresentare una relazione esistente tra un numero di elementi superiore a due, si necessiterebbe di un ODP Logico in grado di rappresentare tale relazione partendo dalle primitive logiche di OWL, che sono in grado solo di utilizzare relazioni binarie e classi. Pur essendo quindi degli strumenti in grado di superare i limiti intriseci di un linguaggio di rappresentazione, sono comunque legati strettamente alle capacità del formalismo scelto.

Alla pagina successiva si può vedere la tabella comprendente tutti i pattern logici proposti finora sul sito per la valutazione da parte del comitato di qualità.

Per questa categoria vengono indicati due tipi di pattern possibili, anche se poi questa divisione non è ufficialmente propagata alle proposte di pattern già presenti sul sito o anche a nuove eventuali proposte: si tratta di una divisione esistente solo nella definizione stessa della categoria, che però riportiamo per senso di completezza dell'opera. Il primo tipo di ODP Logico vengono chiamate "Logical Macro": si tratta di quei pattern che forniscono una rappresentazione ontologica di un'espressione logica ricorrente ed intuitiva: il sito fornisce, come esempio, il costrutto ∇R.C (cioè che ogni R dev'essere un C), che formalmente si esprime come $\exists R.\top \sqcap \forall R.C$, e che in OWL sarebbe espresso come una combinazione di restrizioni owl:allValuesFrom associata a una restrizione owl:someValuesFrom. Un ODP di questo tipo darebbe una "scorciatoia" per passare agilmente dalla forma scritta in formule logiche classiche alla rappresentazione OWL del costrutto.

Nome	Obiettivo
Adrian Walker 2	Integrare la semantica dei dati con la semantica di un metodo d'inferenza, e con i significati di un vocabolario aperto, soprattutto per frasi inglesi a sintassi aperta
Context Slices	Codificare una relazione binaria valida in un contesto specifico
Define Hybrid Class Resolving Disjointness due to Subsumption	Supportare la semantica di una sussunzione definita tra due classi disgiunte e risolvere le conseguenti inconsistenze tramite l'introduzione di una classe ibrida
DisjointnessOfComplements	Definire l'impossibilità di istanze condivise di 2 classi differenti tramite disgiunzione delle due piuttosto che definire le due classi come negazione logica dell'altra
Enlarge Class Definition for Resolving Disjointness due to Subsomption	Supportare la semantica di una sussunzione definita tra due classi disgiunte e risolvere le conseguenti inconsistenze tramite l'espansione della definizione della sotto classe
Summarization of an inverse n-ary relation	Permettere la richiesta di relazione n-arie e le loro inverse esistenti tra due distinte entità senza la necessità di query complesse
N-Ary Relation Pattern (OWL 2)	Permettere l'inferenza sulle proprietà di una relazione tramite reificazione dei componenti della relazioni N-aria originale
NegativePropertyAssertions	Esprimere NPA in ontologie precedenti OWL 2 e dare regole di trasformazione per usare OWL 2
Normalization	Semplificare una poligerarchia, codificando le relazioni di sussunzione attraverso restrizioni piuttosto che creazioni di sottoclassi.
OnlynessIsLoneliness (OIL)	Definire in modo corretto e consistente la relazionabilità delle istanze di una classe con elementi appartenenti a due classi distinte e disgiunte.
Partition	Descrivere come realizzare correttamente una partizione di un insieme
Symmetric n-ary relationship	Permettere la rappresentazione di relazioni simmetriche n-arie (cioè con lo stesso valore per tutti le entità coinvolte nella relazione)
SynonymOrEquivalence (SOE)	Definire in maniera efficiente l'equivalenza tra classi in ontologie distinte ma collegate fra loro

Tabella riassuntiva degli ODP Logici proposti per la valutazione

Il secondo tipo di pattern Logici individuato sul sito è quello il cui scopo è quello di trasformare un'espressione logica da un linguaggio formale ad un altro, come per l'esempio del costrutto logico trasformato in OWL proposto sopra. Per fare ciò però è necessario comunque approssimare l'espressione di partenza, per poter trovare un giusto equilibrio tra requisiti ed espressività per poter rendere possibile la trasformazione senza modificare il significato originale dell'espressione iniziale. Una relazione n-aria non può essere direttamente espressa in OWL: l'approssimazione che rende possibile la rappresentazione di una tale relazione in OWL può essere la creazione di una nuova classe che concretizzi questa relazione, e i cui argomenti vengono indicati come proprietà della nuova classe.

Passiamo ora all'analisi degli ODP finora proposti dai collaboratori del sito.

ALP (Anti-Logical Patterns)

Un gruppo di tre pattern è stato proposto dalle stesse persone, Catherine Roussey e Oscar Corcho, ha introdotto questa definizione. Questi pattern possono essere raggruppati non solo per i comuni autori, ma anche per l'ispirazione comune che hanno avuto: provengono, infatti, tutti dall'esperienza della coppia di studiosi nel dover lavorare con ontologie effettivamente utilizzate al di fuori del mondo accademico. Dopo anni di lavoro, infatti, questi ricercatori dell'Università Politecnica di Madrid hanno notato dei comportamenti errati e ripetuti, attribuibili secondo loro a una scarsa conoscenza teorica delle ontologie da parte dei tecnici addetti alla creazione di ontologie per utilizzo industriale. Da questa loro esperienza sul campo hanno quindi elaborato delle contromisure a quelli che hanno

definito con il termine ALP, o Anti-logical patterns, e le hanno presentate sul sito come pattern Logici.

Andando in ordine alfabetico, il primo di questo gruppo è il pattern intitolato "DisjointnessOfComplement (DOC)". Il pattern è molto semplice: il loro lavoro di debugging su ontologie "reali" li ha portati a notare che, quando lo sviluppatore intendeva evidenziare che due classi C1 e C2 non potevano avere istanze condivise, definiva una come la negazione logica dell'altra, e questo portava a errori di modellizzazione o a classi senza istanze perché avevano requisiti insoddisfacibili. Per contrastare questo anti-pattern quindi hanno proposto la semplice sostituzione della troppo utilizzata negazione logica tra classi con la più corretta ed efficiente disgiunzione delle classi: in pratica, si riduce alla semplice trasformazione da "C1 isEquivalentTo not C2" a "C1 disjointWith C2".

L'ODP successivo proposto da questi ricercatori ha nome "OnlynessIsLoneliness (OIL)": in questo pattern si focalizza l'attenzione su un altro problema generato da uno scorretto uso e/o interpretazione delle regole delle logiche descrittive. L'errore abituale, contrastato da questo pattern, si verificava quando lo sviluppatore aggiungeva all'ontologia una restrizione del tipo "le istanze di C1 sono relazionabili solo con le istanze di C2 tramite la proprietà R" e contemporaneamente (o successivamente; in ogni caso le due restrizioni erano presenti allo stesso momento nella knowledge base) un'altra restrizione identica, ma che al posto di C2 aveva una terza classe C3, disgiunta da C2. Ovviamente ciò causa delle situazioni impossibili da soddisfare, ma nonostante l'evidenza della cosa, questo comportamento, assicurano gli autori, è molto diffuso nelle ontologie usate in pratica nel mondo. Viene proposta quindi come soluzione la seguente formalizzazione: al posto di avere le tre regole "C1 subClassOf R only C2", "C1 subClassOf only C3" e " C2 disjointWith

C3", si utilizzi la regola più corretta e sicuramente soddisfacibile "C1 subClassOf R only (C2 or C3)", accompagnata da "C2 disjointWith C3".

Il terzo e ultimo pattern proposto da questi autori si intitola "SynonymOrEquivalence (SOE)": in questo caso si cerca di porre rimedio alla pratica diffusa di indicare due classi equivalenti con nomi diverse, in quanto provenienti da due ontologie separate, con la semplice sostituzione delle istanze della classe meno usata con istanze dell'altra classe. In sostanza, secondo gli autori, spesso gli sviluppatori di ontologie creano due classi distinte quando in realtà concettualmente sarebbe più corretto creare un'unica classe con due diverse etichette, soprattutto perché frequentemente una delle due classi create è superflua in quanto non viene praticamente mai riutilizzata all'interno della stessa ontologia. Il pattern, anche in questo caso piuttosto semplice a livello di implementazione, propone l'eliminazione della classe meno usata e il trasferimento di tutti i suoi commenti ed etichette alla classe equivalente presente nell'ontologia.

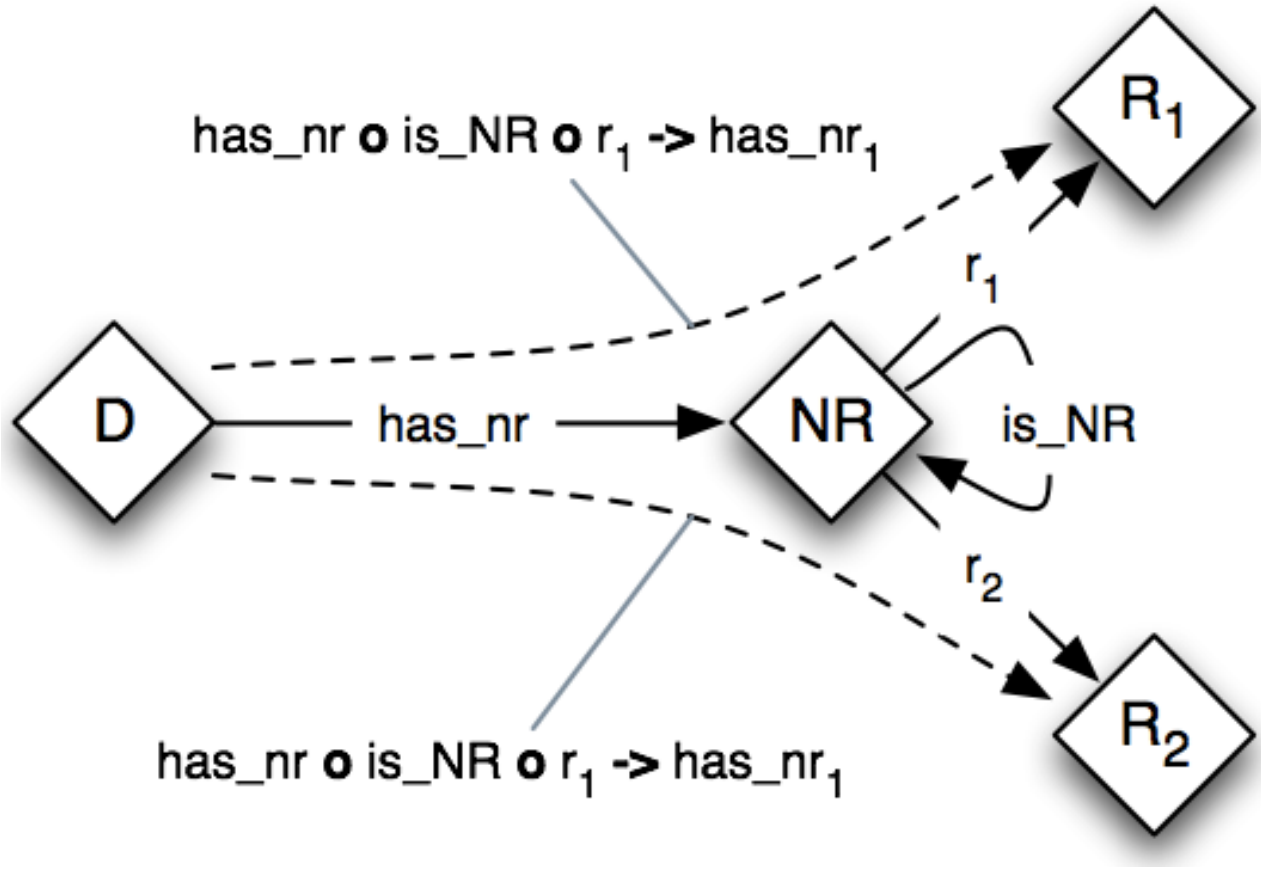
ODP Logici su relazioni tra entità

Un altro piccolo sottogruppo individuabile all'interno della categoria degli ODP Logici è quello riguardante le relazioni esistenti fra entità di un'ontologia. E' composto da tre pattern, due proposti da Maria Poveda e Mari Carmen Suarez Figueroa, entrambi ricercatori presso l'Università Politecnica di Madrid, e uno proposto da Rinke Hoekstra, ricercatore dell'università di Amsterdam.

Quest'ultimo ha proposto un pattern per l'analisi da parte del comitato di qualità su cui si sono basate le ricercatrici spagnole per ideare i loro ODP, per cui partiamo con l'analisi di questo pattern: si intitola semplicemente "N-ary Relation Pattern (OWL2)", e lo scopo

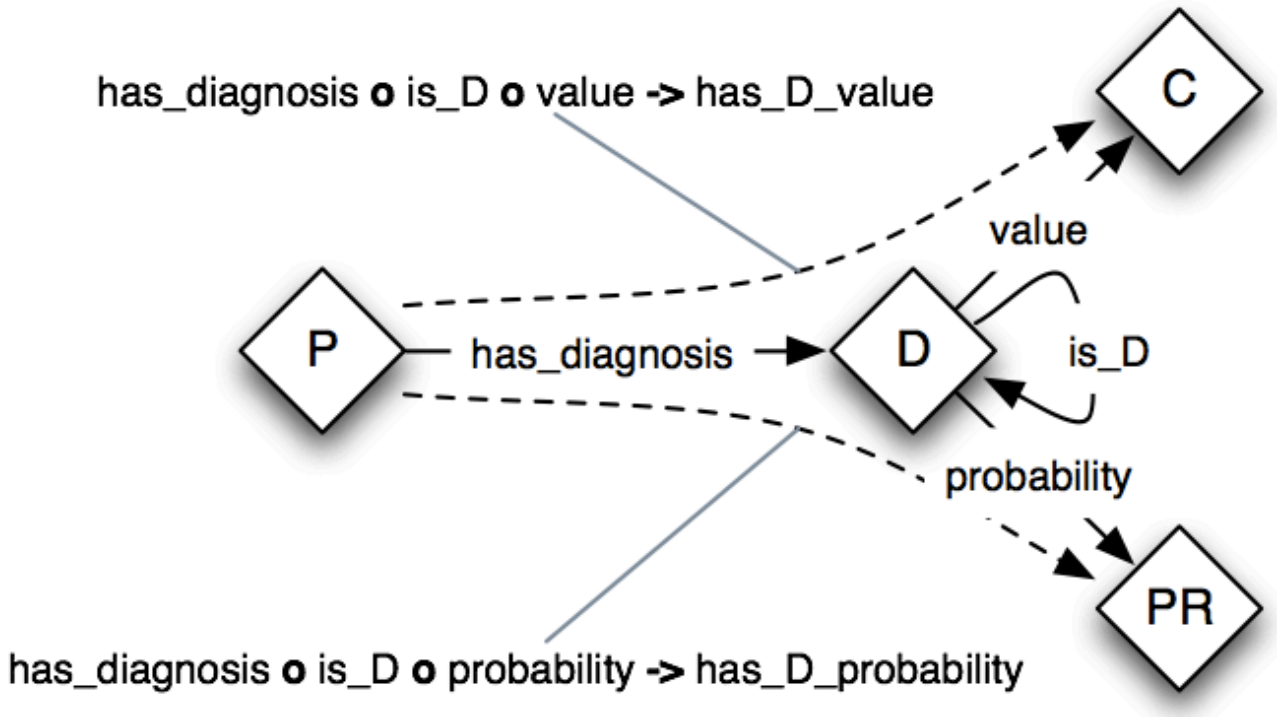
intuitivo di questa proposta è quello di fornire uno schema per poter creare relazioni n-arie in OWL tramite la reificazione delle stesse ma permettendo allo stesso di tempo di eseguire delle inferenze tra i diversi partecipanti della relazione nonostante con la reificazione si vada a perdere la caratteristica e tradizionale struttura delle relazioni tra entità ontologiche.

Quindi, per realizzare le relazioni n-arie non supportate da OWL si procede in questo modo: si crea una nuova classe rappresentante la relazione (la classe NR nello schema seguente), mentre per ogni possibile argomento ulteriore viene creata una proprietà (r_1 e r_2 nell'esempio) a cui vengono assegnati i valori rispettivi.



Schema generico di una relazione n-aria reificata

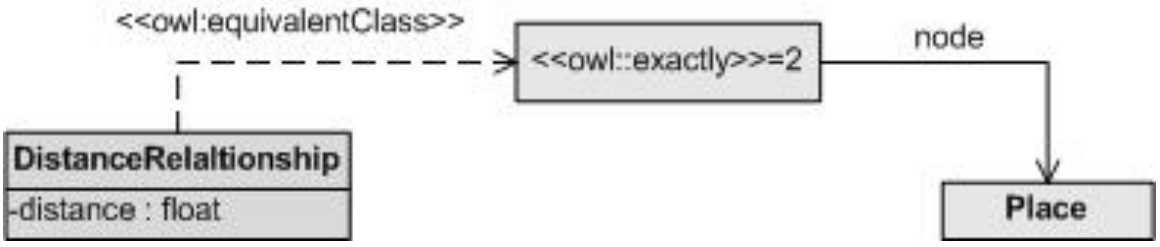
Per meglio chiarire, alleghiamo anche l'immagine che gli autori usano come esempio: lo schema seguente rappresenta la relazione "il paziente P (Christine) ha una diagnosi di cancro (C) con probabilità PR", dove D rappresenta la relazione reificata "ha diagnosi".



Esempio di relazione n-aria reificata

Passiamo ora al secondo di questi pattern, che, come suggerisce il titolo "Symmetric n-ary relationship", si occupa di dare una descrizione di come realizzare delle relazione n-arie simmetriche all'interno di una ontologia. Lo spunto per questo pattern, come spiegano gli autori, è venuto dalla semplice necessità di modellare la distanza euclidea tra due punti, che probabilmente è l'esempio più semplice di relazione simmetrica. Si ha infatti la necessità di rappresentare nella relazione sia entrambi i punti coinvolti sia il valore stesso della distanza tra i due, e che ovviamente il valore sia identico nella relazione nella sua inversa.

La soluzione proposta da questo pattern è facilmente rappresentabile con un immagine:

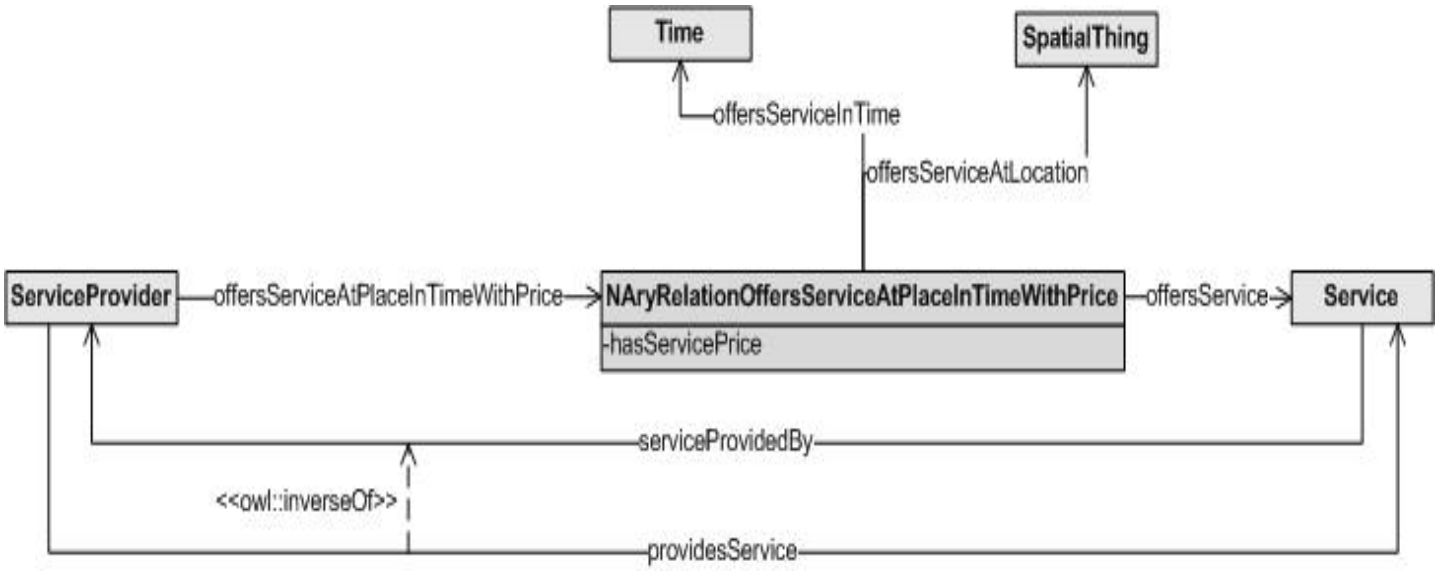


Com'è intuibile, la relazione esistente tra le due entità viene rappresentata come una classe a sé stante, e le viene assegnata una lista di proprietà, una per ciascun elemento successivo ai primi due necessario alla modellazione della relazione. Nell'esempio infatti si nota come la relazione di distanza sia rappresentata da una classe, a cui è assegnato una proprietà dal valore 2 (che rappresenta, appunto, il valore numerico della distanza tra i due punti).

Il secondo pattern elaborato dalle ricercatrici spagnole collegato alle relazioni tra entità, è praticamente il simmetrico del precedente: intitolato "Summarization of an inverse n-ary relation", si pone l'obiettivo di permettere la creazione di relazione n-arie e delle loro inverse senza dover ricorrere necessariamente a complicate query. Lo spunto per il pattern è nato da due diverse necessità: la prima, la più ovvia, la necessità di realizzare un modello per l'inversione di relazioni n-arie con partecipanti distinti; la seconda riguarda la creazione di una scorciatoia per permettere agli utenti di poter formulare query riguardanti i componenti di una relazione n-aria. Bisogna però notare che la struttura suggerita in questo pattern sottintende che all'interno della relazione ci siano due entità coinvolte che hanno un ruolo maggiore: come nell'ODP precedente, i componenti ulteriori ai primi due vengono realizzati tramite l'aggiunta di proprietà a una classe creata apposta per rappresentare la relazione, creando così una sorta di gerarchia all'interno della stessa.

Come fanno notare gli autori, questo può anche portare ad avere in maggior risalto tra gli altri un singolo partecipante, che potrebbe perfino essere indicato come “proprietario” della relazione stessa.

Riportiamo ora l’immagine che gli autori usano come esempio pratico di applicazione del pattern: lo schema rappresenta la relazione esistente tra un fornitore di servizi e il servizio stesso, in cui, oltre ai due ovvi partecipanti, si è voluto specificare anche gli “orari di apertura” e l’ubicazione del probabile negozio in cui viene fornito il servizio. Come si può vedere, oltre alle relazioni binarie classiche tra il fornitore e il servizio, esiste anche la classe rappresentante la relazione n-aria tra i due partecipanti principali e gli altri (si può facilmente notare come effettivamente con questa struttura si venga a creare una gerarchia tra i componenti della relazione). Grazie proprio all’esistenza di questa relazione n-aria, è possibile per gli utenti creare query semplici per poter risalire ai componenti “secondari” della relazione.

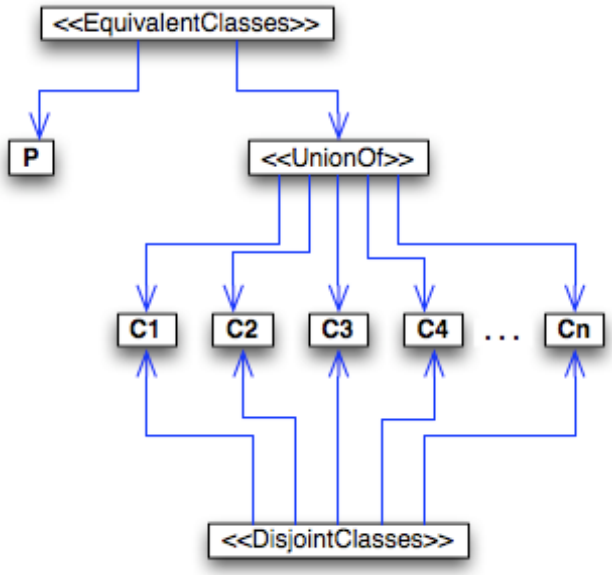


Esempio di relazione n-aria con inversa

Altri ODP Logici

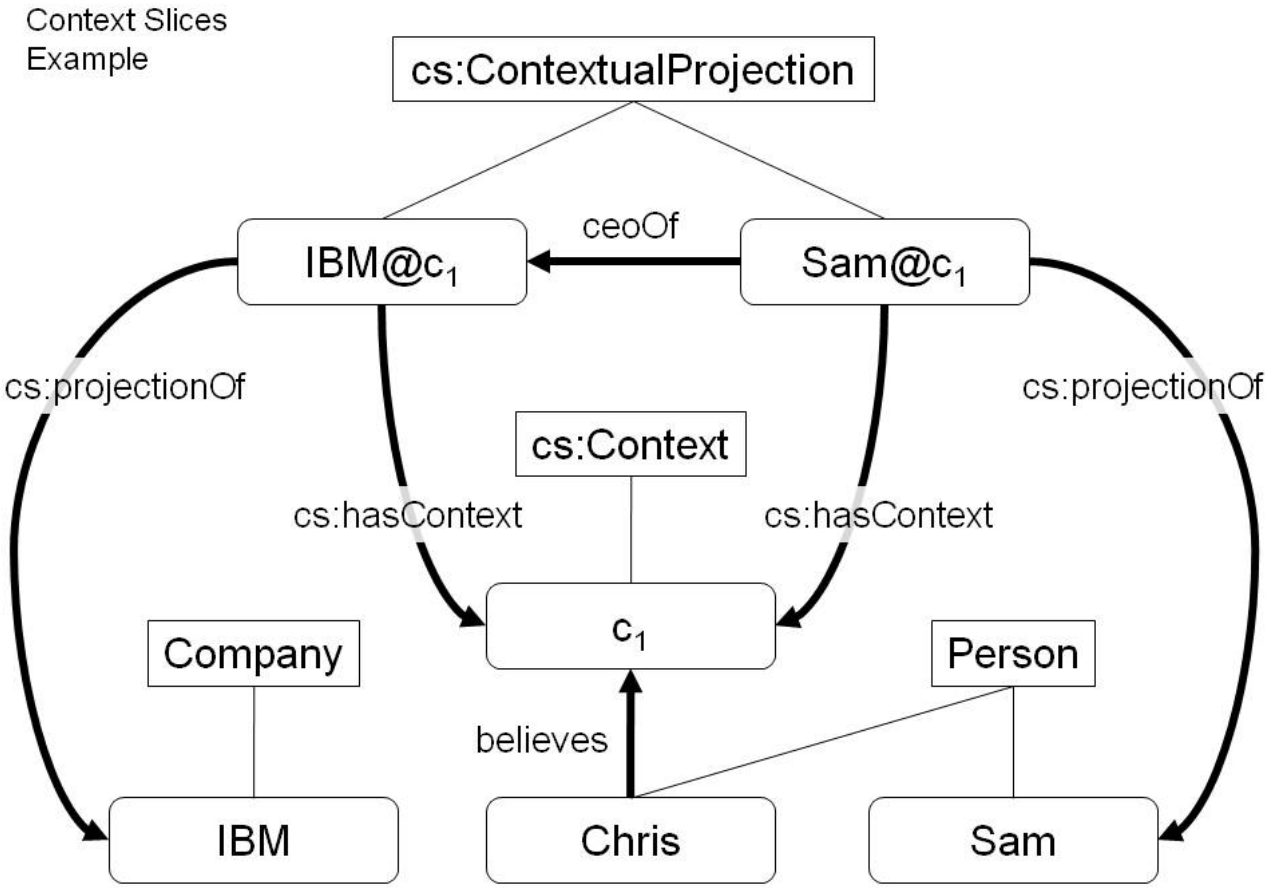
Gli altri ODP Logici presenti sul sito non possono essere categorizzati sotto un qualche aspetto comune, per cui ne presentiamo solo alcuni che abbiamo ritenuto maggiormente interessanti per la nostra ricerca.

Uno di questi pattern, veramente molto semplice e intuitivo ma ciò non di meno utile, è intitolato “Partition”, e tratta a livello ontologico il concetto ben conosciuto di partizione: quando un insieme di elementi è diviso in sottogruppi distinti che, oltre a non avere elementi in comuni, non lasciano escluso nessun elemento dell’insieme originario. Un esempio rapido e intuitivo è la suddivisione dei numeri naturali in pari e dispari: questa partizione in due gruppi comprende tutti gli elementi dell’insieme di partenza, nessuno escluso, e contemporaneamente non esiste un elemento che appartenga a più di un sottogruppo della partizione applicata. Il pattern viene quindi realizzato ontologicamente come una semplice lista di classi disgiunte tra loro, e la loro unione viene definita come equivalente al gruppo originale a cui viene applicata la partizione.



Schema generico dell’ODP “Partition”

Passiamo ora un altro ODP, e analizziamo quello intitolato “Context Slices”, il cui obiettivo è quello di fornire un metodo efficiente e rapido per rappresentare credenze e convinzioni in una precisa contestualizzazione, intesa sia come ciò che può credere una persona o un entità, sia come informazioni vere solo per un certo periodo di tempo o, ancora, come ciò che viene riferito/testimoniato da un individuo. Gli autori hanno realizzato questo pattern alla luce della grande quantità di informazioni presenti su internet che necessitano questo tipo di contestualizzazione per essere utilizzate in maniera corretta, e al posto di usare la più diffusa reificazione delle relazioni (che portava a una perdita della potenza espressiva del linguaggio usato per codificarle), hanno pensato di creare una “proiezione” dei partecipanti alla relazione che esprime il credo in questione. Nello schema seguente, rappresentante l’esempio fornito dagli autori assieme alla struttura del pattern, si può notare come venga rappresentata ontologicamente la frase “Chris believes that Sam is CEO of IBM”: i due attori coinvolti nella convinzione di Chris, Sam e IBM, vengono “duplicati” e i loro doppi vengono legati a Chris tramite il contesto C1, che rappresenta la reificazione di “ciò che crede Chris”. Tutta l’informazione viene quindi rappresentata tramite relazione tra le classi contestualizzate dei partecipanti, escluso il “credente”: la relazione *ceoOf* coinvolge solo le proiezioni contestualizzate delle entità rappresentanti gli attori, e ciò permette quindi di evidenziare come questa informazione possa perfino non essere vera nonostante sia rappresentata nell’ontologia (o essere in qualche altro modo dipendente dal contesto). Da notare come questo pattern permetta la rappresentazione di più informazioni derivanti da contesti diversi: se per esempio volessimo rappresentare la convinzione di Sam che Chris sia il CEO di IBM, basterebbe collegare a Sam un secondo

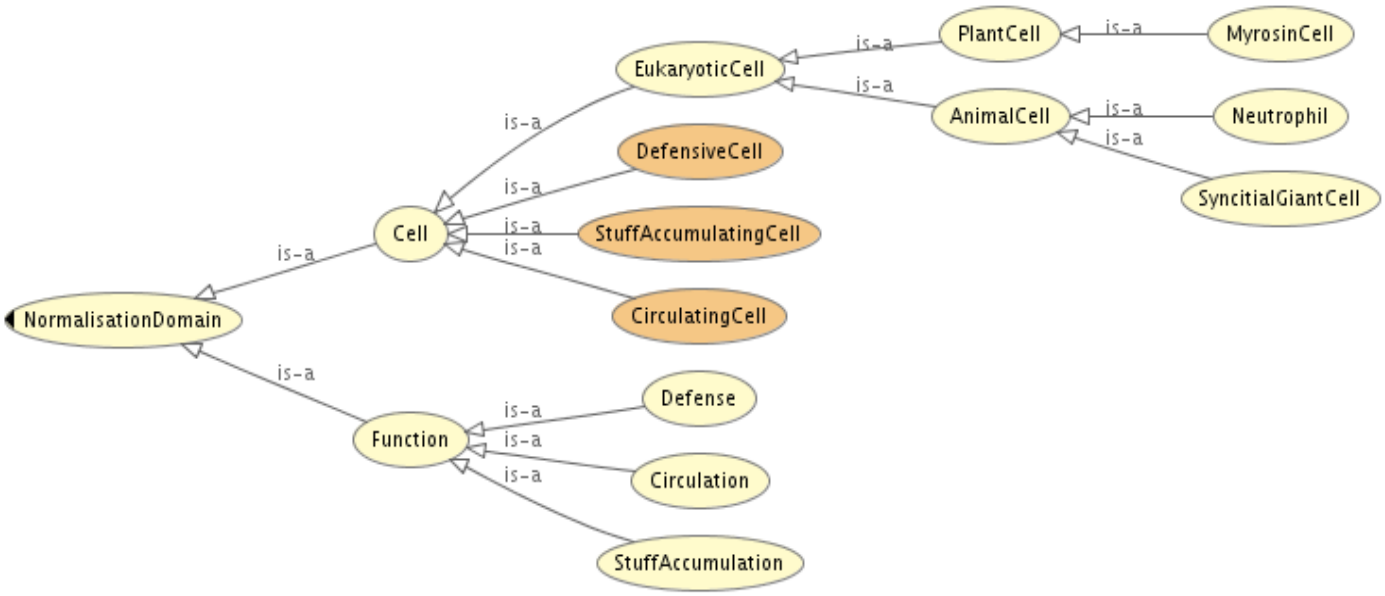


Esempio di applicazione dell'ODP "Context Slices"

contesto C2, a cui farebbero riferimento due distinte proiezioni contestuali delle classi Chris e IBM collegate tra loro dalla stessa relazione *ceoOf* . Questo ODP quindi permetterebbe di rappresentare queste due idee anche se, per ipotesi, la classe IBM permettesse l'esistenza di un solo CEO (che potrebbe non essere né Sam né Chris). Il pattern di per sé è molto semplice: in fondo gli autori si sono limitati ad aggiungere due classi, *cs:Context* e *cs:ContextualProjection* che permettono però di esprimere concetti e ontologie anche molto complesse.

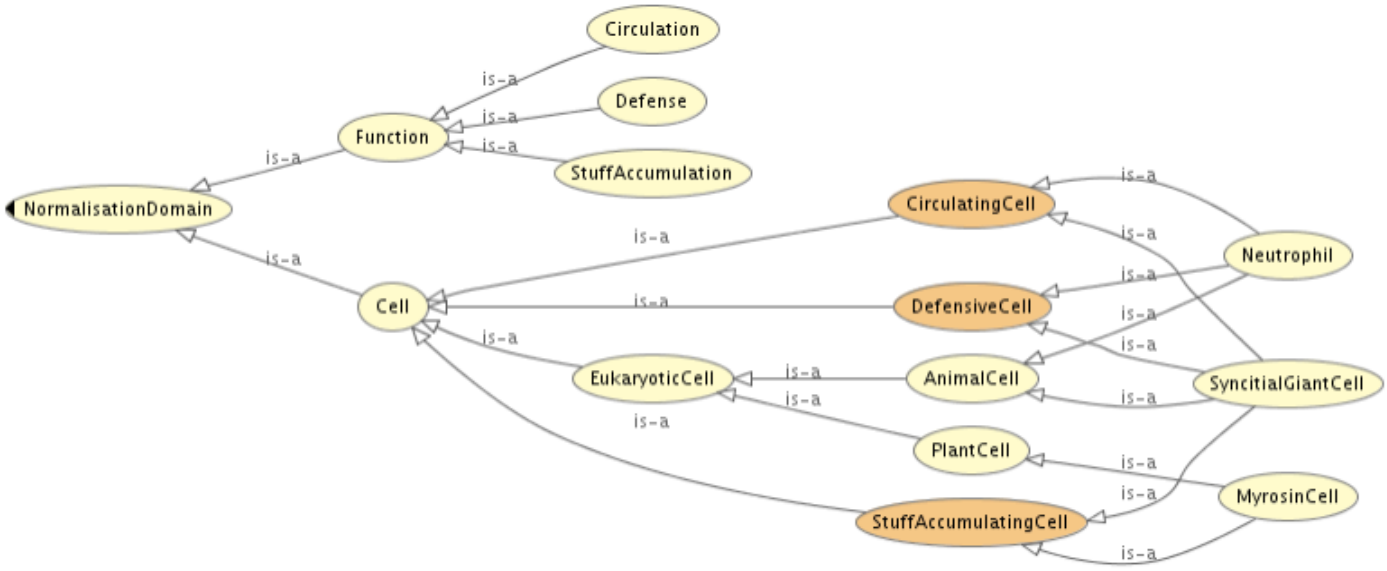
L'ultimo pattern della categoria che andiamo ad esaminare è lievemente più complesso dei precedenti: si intitola "Normalization", e tenta di fornire uno schema utile per la

rappresentazione ontologica di situazioni in cui si hanno classi con molte e diverse superclassi a cui si riferiscono (viene spesso chiamata poligerarchia una simile struttura). Spesso questa situazione viene implementata tramite sussunzione, ma ciò porta generalmente ad avere due problemi: in primo luogo l'ontologia diventa difficile da mantenere, in quanto la modifica o l'aggiunta di nuove classi può coinvolgere un gran numero di assiomi e di proprietà da correggere, con un'alta probabilità di inserire errori o di ottenere risultati diversi da quelli desiderati; come secondo problema, più a livello teorico, si ha che le relazioni semantiche sono implicite, e ciò non permette a un utente o chiunque non sia il creatore dell'ontologia di sapere perché una classe sia sottoclasse di un'altra. Gli autori del pattern hanno ideato quindi questa soluzione: al posto di definire le sussunzioni tramite le classiche relazioni classe-sottoclasse, usano le più semplici e maneggevoli restrizioni di dominio. Vediamo un esempio per meglio chiarire il funzionamento di questo pattern. Nello schema alla pagina seguente viene rappresentata una poligerarchia riguardante una cellula e le sue diverse tipologie possibili, realizzata tramite le consuete relazioni classe-sottoclasse. Si vede come quindi le cellule siano divise sia per tipo sia per funzione: gli autori portano come esempio la classe *Neutrophil*, che, essendo una sottoclasse di *Cell*, è quindi anche sottoclasse di *DefensiveCell*, *StuffAccumulatingCell* e *CirculatingCell*. Questo, anche se non è visibile dallo schema, è il "groviglio", come lo chiamano gli autori, di rapporti di classe-sottoclasse presenti e difficilmente maneggiabili.



Schema esemplificativo di una poligerarchia "classica"

Nello schema successivo si ha invece la stessa ontologia rappresentata utilizzando il pattern proposto: anche se dal punto di vista ottico può apparire meno chiara e comprensibile, è invece sicuramente più chiara e efficiente dal punto di vista ontologico. Nonostante gli incroci grafici nella rappresentazione, sono comunque più chiari i rapporti di "parentela" tra le classi e le loro sottoclassi: con la moltiplicazione delle relazioni tra



Schema esemplificativo della stessa poligerarchia ma creata con l'ODP "Normalization"

esse, si nota subito come la stessa classe *Neutrophil* di prima sia “figlia” delle tre specializzazioni di *Cell*, e di conseguenza della classe *Cell* stessa.

Capitolo 5 – Nuove proposte di ODP Logici

In questo capitolo vengono quindi proposti nuovi pattern logici, strutturati secondo i requisiti richiesti per la pubblicazione e l'eventuale revisione da parte del comitato di qualità. Si è scelto di proporre solo nuovi pattern per la categoria ODP Logici per le stesse motivazioni portate all'inizio del capitolo precedente.

Vediamo ora i pattern che si è deciso di pubblicare sul sito. Il primo di questi pattern è stato nominato "Specialization through attributes", e come si può intuire riguarda la creazione di sottoclassi attraverso l'aggiunta di nuovi attributi alla classe già esistente. Per fare un esempio pratico, si pensi di voler creare una sottoclasse *Ingegnere* della classe *Laureato*: per sottolineare la differenza tra le due, bisogna aggiungere degli ulteriori ruoli alla classe *Laureato*, come ad esempio un ruolo che ne rappresenti l'iscrizione all'albo o che comunque ne rappresenti la promozione all'esame di stato, e sarebbe opportuno anche aggiungere una proprietà che valuti il conseguimento della laurea presso un ateneo ingegneristico. Gli assiomi che rappresentano questo pattern sono rappresentati nella tabella alla pagina seguente, e si è scelto di rappresentarli col linguaggio delle logiche descrittive per mantenerli il più possibile astratti e separati da un linguaggio di programmazione specifico.

Assioma DL	Descrizione
$B \equiv A \sqcap \exists R_1 \sqcap \exists R_2 \sqcap \exists R_3 \ldots \sqcap \exists R_n$	La classe B è definita come sottoinsieme della classe A aggiungendo dei ruoli che specificano attributi che costituiscono le differenze dalla classe dalla cui si sta specializzando.
$R_1 : B \rightarrow E_1$ $R_2 : B \rightarrow E_2$ $R_3 : B \rightarrow E_3$ … $R_n : B \rightarrow E_n$	Definizione degli n ruoli che rappresentano tutti gli attributi da aggiungere per la specializzazione che si vuole applicare. Alcuni di questi ruoli possono esser anche già presenti nella classe di partenza. Più ruoli possono riferirsi alla stessa classe Ex, se è necessario.
$A \sqsubseteq \leqslant a\ R_1$ $A \sqsubseteq \leqslant b\ R_2$ … $A \sqsubseteq \leqslant n\ R_n$	Con questi assiomi si possono specificare quantitativi massimi o minimi di attributi che la specializzazione in esame consente, tenendo però presente che comunque le costanti a,b,…n devono aver un valore minimo di 1, in quanto devono essere delle aggiunte alla classe di origine.

Tabella con gli assiomi DL per il pattern "Specialization through attributes"

L'esempio successivo che si è elaborato riguarda ancora la specializzazione di una classe, ma il meccanismo scelto in questo caso per la creazione di nuove sottoclassi è la restrizione del codominio. Il pattern è stato quindi denominato "Specialization through codomain restriction", e per dare un esempio pratico di applicazione rimaniamo sullo stesso tema dell'esempio del precedente pattern: ammettendo che si voglia creare un'ulteriore specializzazione della classe *Ingegnere*, sarebbe possibile realizzare le sottoclassi *IngegnereInformatico* o *IngegnereChimico* applicando, per l'appunto, una restrizione al codominio della proprietà *LaureatoIn* che limiti i possibili valori a "Ingegneria Informatica" o "Ingegneria Chimica" rispettivamente; perché sia una vera restrizione del codominio ovviamente la proprietà deve essere una già presente nella definizione della classe che si vuole specializzare, altrimenti si ricade nel caso del pattern precedente, in cui

si ottengono le sottoclassi per aggiunta di attributi. Nella tabella seguente sono presentati gli assiomi in logica descrittiva che rappresentano questo tipo di specializzazione.

Il pattern successivo che si è elaborato per la pubblicazione sul sito riguarda invece una struttura che coinvolge molti ambiti, non solo quelli strettamente legati all'informatica o alle ontologie: si è infatti elaborata una serie di assiomi in grado di esprimere una gerarchia ad albero, con la specifica però che il padre di ciascuna classe può essere uno solo (e non, come nel caso visto tra gli ODP già proposti, di una "poligerarchia"); il pattern è stato chiamato semplicemente "Tree Classification". Questa struttura è sicuramente molto diffusa e conosciuta non solo nell'ambito del software: può rappresentare infatti sia organigrammi aziendali sia, per fare un esempio, la classificazione dei diversi tipi di metalli presenti nella tavola periodica.

Assioma DL	**Descrizione**
$R_1 : A \rightarrow C_1$ $R_2 : A \rightarrow C_2$ $R_3 : A \rightarrow C_3$ … $R_n : A \rightarrow C_n$	Elenco di ruoli che definiscono una serie di attributi già presenti nella classe A che si vuole specializzare, e necessariamente obbligatori per appartenere alla suddetta classe.
$A \sqsubseteq \leqslant a\ R_1$ $A \sqsubseteq \leqslant b\ R_2$ … $A \sqsubseteq \leqslant n\ R_n$	Qui indichiamo gli eventuali vincoli presenti su ciascun ruolo nella classe d'origine A.
$B \equiv A \sqcap$ $\exists\ R_A.\{a\} \sqcap$ $\forall\ R_B.\{b\} \sqcap$ $\leqslant x\ R_C.\{c\}$ …	Con questi assiomi si dichiara infine la specializzazione che si vuole attuare, indicando per ogni ruolo di cui si vuole restringere il codominio il tipo di quantificatore che si vuole utilizzare e se necessario anche la qualificazione richiesta dalla specializzazione.

Tabella con gli assiomi DL per il pattern "Specialization through codomain restriction"

Assioma DL	Descrizione
$A \equiv B_1 \sqcup B_2 \sqcup B_3 \sqcup ... \sqcup B_N$	Nodi o foglie direttamente discendenti dalla radice, di prima generazione.
$B_1 \equiv C_1 \sqcup C_2 \sqcup C_3 \sqcup ... \sqcup C_N$ $B_2 \equiv D_1 \sqcup D_2 \sqcup D_3 \sqcup ... \sqcup D_N$ … $B_N \equiv Z_1 \sqcup Z_2 \sqcup Z_3 \sqcup ... \sqcup Z_N$	Nodi o foglie di seconda generazione, discendenti da un nodo di prima generazione, e così via fino alla n-esima generazione
Disjoint (C_2, D_2,D_3, … Z_N)	Mettendo in questo assioma TUTTE le foglie senza "figli" della classificazione, le si dichiara disgiunte, e si può quindi dedurre la disgiunzione anche di tutti i nodi sovrastanti le foglie.

Tabella con gli assiomi DL per il pattern "Tree Classification"

La tabella precedente illustrava gli assiomi necessari a rappresentare questa comune struttura ad albero.

Il pattern successivo è intitolato "Attributes Classification", e si pone lo scopo di rappresentare quelle classificazioni per cui non è ottimale utilizzare una struttura ad albero in quanto non è presente nessuna notevole distinzione gerarchica tra le varie caratteristiche che si vogliono evidenziare. Un esempio ormai classico è quello della classificazione attuabile tra i diversi tipi di vino: non essendoci una particolare gerarchia in cui suddividere le diverse caratteristiche dei vini (rosso/bianco, spumante/liscio, ecc.), la classificazione più corretta che si può fare è quella rappresentabile tramite un grafico cartesiano con un numero di dimensioni pari al numero di attributi possibili. Per esempio, per i vini si otterrebbe una classificazione tridimensionale rappresentando su un asse il colore del vino, su un altro la gradazione e su un altro ancora il tipo di vitigno usato. Come si può notare, è difficile stabilire una gerarchia tra queste caratteristiche che rispecchi

davvero la realtà: se si volesse usare una rappresentazione ad albero, sarebbe comunque fattibile, ma la struttura scelta a quel punto sarebbe probabilmente arbitraria e non rispondente a veri criteri gerarchici. La tabella seguente, come al solito, rappresenta gli assiomi in logica descrittiva utili a ottenere questo tipo di classificazione.

Assioma DL	**Descrizione**
$FEAT_1 \equiv \{ Value_{11}, Value_{12}, Value_{13} \dots Value_{1N} \}$ $FEAT_2 \equiv \{ Value_{21}, Value_{22}, Value_{23} \dots Value_{2N} \}$ ... $FEAT_M \equiv \{ Value_{M1}, Value_{M2}, Value_{M3} \dots Value_{MN} \}$	Elenco degli attributi che rappresentano la categoria di classi che si vuole descrivere tramite questa classificazione.
$\neq (Value_{11}, Value_{12} \dots Value_{1N}, Value_{21}, Value_{22} \dots Value_{MN})$	Si differenziano tutti gli individui che rappresentano un attributo, così da evitare equivoci o inconsistenze.
$R_1 : C \rightarrow FEAT_1$ $R_2 : C \rightarrow FEAT_2$ ... $R_M : C \rightarrow FEAT_M$	Ogni ruolo corrisponde a una caratteristica posseduta dall'insieme di individui che si vuole descrivere.
$C \sqsubseteq =1\ R_1 \sqsubseteq =1\ R_2 \sqsubseteq =1\ R_3 \sqsubseteq \dots =1\ R_M$	L'insieme degli individui che si vuole descrivere è definito da tutti quelli che hanno esattamente una caratteristica per ogni attributo che si è precedentemente definito.
$C\text{-}Value_{11} \equiv \exists\ R_1.\{Value_{11}\}$	Si possono inoltre definire sottoclassi di individui che condividono una o più caratteristiche, utilizzando assiomi come questo.

Tabella con gli assiomi DL per il pattern "Attributes Classification"

L'ultimo pattern che si è elaborato per la pubblicazione sul sito è denominato "Part of", e come lascia intendere il nome, riguarda la relazione "A è costituente/parte di B", intesa come il rapporto che ci può essere tra un individuo qualunque e un insieme di cui esso fa parte, che può avere altri ruoli oltre alla semplice appartenenza dell'individuo all'insieme, e con eventuali restrizioni sulla quantità di questi ruoli. Un esempio concreto di questo tipo di relazione può essere quello che descrive l'appartenenza di un azionista alla società di cui possiede una parte, oppure la relazione tra un componente di un utensile e l'utensile stesso, come il motore di una macchina e la macchina stessa.

Assioma DL	**Descrizione**
$C \sqcap D \equiv \bot$	I due termini sono definiti come disgiunti
$R1 : C \rightarrow D$	Proprietà "base" di appartenenza di D a C, non è obbligatorio.
$R2 : C \rightarrow D$ $R3 : C \rightarrow D$ $R4 : C \rightarrow D$ … $Rx : C \rightarrow D$	Proprietà accessorie che evidenziano compiti/ruoli diversi all'interno del "tutto", possono esser quanti si vuole.
$C \sqsubseteq \leqslant a\ R1 \sqcap$ $\leqslant b\ R2 \sqcap$ $\leqslant c\ R3 \sqcap$ … $\sqcap \leqslant n\ Rx$	Questi enunciati servono a definire il "tutto" tramite i compiti/ruoli che sono ritenuti necessari per l'esistenza del "tutto" stesso. Vanno indicati in questo modo tutti i compiti/ruoli che sono prerequisiti per poter considerare il "tutto" come tale.
$R2 \sqsubseteq R1$ $R3 \sqsubseteq R2$ $R4 \sqsubseteq R2$ $R5 \sqsubseteq R3$ … $Rx \sqsubseteq Ry$	Qui si indicano i vari rapporti di inclusione che sussistono fra i compiti/ruoli, evidenziando così eventuali sottoinsiemi e dando un'idea approssimativa della gerarchia all'interno del "tutto".
$Rx \sqcap Ry \equiv \bot$	Si possono poi aggiungere enunciati di questo tipo per specificare come due compiti/ruoli non possano esser ricoperti dallo stesso individuo.

Tabella con gli assiomi DL per il pattern "Part Of"

Capitolo 6 – Conclusioni

Il lavoro di ricerca svolto finora mostra come l'applicazione dei design pattern nelle ontologie sia ancora un campo in via di sviluppo: a livello accademico si stanno muovendo sicuramente importanti passi verso la creazione per un repository di pattern standard da cui studiosi e progettisti potranno attingere per poter meglio creare basi di conoscenza; a livello di applicazione pratica, i cosiddetti "antipattern" individuati dal duo Roussey-Corcho ci mostrano come nelle ontologie usate in pratica siano più frequenti pattern, per l'appunto, negativi senza però parlare della presenza di design pattern veri e propri diffusi tra i progettisti di ontologie.

I prossimi passi della ricerca sono legati al sito www.ontologydesignpattern.org a doppio filo: da un lato sia per chi ha design pattern da proporre e cerca un mezzo per poterli diffondere al più ampio pubblico disponibile, dall'altro sia per chi desidera contribuire alla creazione di un ente *super partes* che possa certificare la qualità dei design pattern come lo è il W3C per l'HTML. E' sicuramente un compito che richiederà molto tempo, ciò che il professor Gangemi e gli altri componenti del comitato editoriale e di qualità che si occupano di gestire il sito e di valutare i design pattern proposti dalla collettività degli utenti: ciò non di meno riteniamo che sia un progetto necessario e molto utile per favorire la diffusione e la conoscenza degli ODP, e quindi per poter permettere la creazione da

parte dei progettisti di ontologie sempre più efficienti e standardizzate. La scelta di usare un sistema “wiki” per quanto riguarda la gestione degli utenti e la possibilità degli stessi di contribuire con il proprio bagaglio di esperienza personale sicuramente è ottimale per spingere i ricercatori sparsi per il mondo a fornire il proprio contributo a quello che si prospetta essere un lavoro che, comunque vada, richiederà sicuramente tempo.

Bibliografia

- Gamma et Al.: Design Patterns: *Elements of Reusable Object-Oriented* , 1995.
- Meyer, B.: *Object-Oriented Software Construction* (Book/CD-ROM) (2nd Edition), Prentice Hall PTR, March 2000. *[architectural ODP example]*
- Rector, A.L.: "Modularisation of domain ontologies implemented in description logics and related formalisms including OWL", in: *K-CAP '03: Proceedings of the 2nd international conference on Knowledge capture*, New York, NY, USA, ACM 2003. *[architectural ODP example, logical ODP: Normalization]*
- Welty, Chris and Richard E. Fikes. E.: "A Reusable Ontology for Fluents in OWL", in Bennet and Fellbaum: *Proceedings of the Fourth International Conference on Formal Ontology in Information Systems*, IOS Press, 2006. *[logic ODP:Context Slices]*
- Patel-Schneider, P. F., Swartout, B.: *Description-Logic Knowledge Representation System Specification*, 1993. *[logical ODP: Partition]*
- Motik, B., Patel-Schneider, P. F., Parsia, B.: *OWL 2 Structural Specification and Functional-Style Syntax*, 2009. *[logical ODP: Partition]*
- Denton, W.: *How to Make a Faceted Classification and Put It On the Web Description*, Nov 2003. *[reenginering ODP: CyclicSubClassOf]J*
- Dearden, A. and Finlay, J.: *Pattern Languages in HCI: A critical review*, Jan 2006.
- Beck, K. and Cunningham, W.: *Using Pattern language for object-oriented programming*, 1987
- http://www.bioinfo.de/isb/gcb99/talks/reich/main.html