POLITECNICO DI MILANO

Scuola di Ingegneria Industriale e dell'Informazione
Corso di Laurea Magistrale in Ingegneria Informatica

Un'ontologia OWL per monitorare il ciclo di vita di obblighi e divieti

Relatore: Prof. Marco Colombetti

Tesi di Laurea di:

Dario Federico Porzio, matricola 770832

Anno Accademico 2012-2013

SOMMARIO

La definizione, la gestione e il monitoraggio di obblighi e divieti con intervalli di validità e condizioni di attivazione e terminazione sono attività importanti nello sviluppo di sistemi distribuiti, in cui è necessario rappresentare le interazioni tra agenti e le conseguenze delle loro azioni.

In questa tesi realizzeremo un modello adatto a tali scopi tramite il linguaggio OWL e regole SWRL. Il modello descrive le proprietà formali delle relazioni deontiche, degli eventi ai quali si riferiscono e degli stati assunti durante il loro ciclo di vita. Successivamente descriveremo come, attraverso la definizione di una semplice applicazione Java, sono gestiti gli eventi che modificano lo stato di obblighi e divieti sfruttando le potenzialità del reasoner e mostreremo un esempio di utilizzo di questo modello.

SUMMARY

The specification, management and monitoring of obligations and prohibitions with validity intervals and conditions of activation and termination are crucial aspects in the development of distributed systems in which it is necessary to represent the interactions between agents and the consequences of their actions.

In this thesis we propose a suitable model for such purposes using OWL and SWRL rules. The model describes the formal properties of the deontic relations, the events to which they relate and the states assumed during their life cycle. Later we will describe how are managed events that change the status of obligations and prohibitions exploiting the potential of the reasoner, through the definition of a simple Java application, with a realistic example.

RINGRAZIAMENTI

Desidero ringraziare tutti coloro che mi hanno aiutato nella stesura della tesi con suggerimenti, critiche ed osservazioni.

Ringrazio anzitutto il professor Marco Colombetti per il suo supporto durante tutte le fasi del progetto, per la sua pazienza, la sua comprensione e la sua gentilezza.

Ringrazio i miei colleghi per avermi aiutato, concedendomi del tempo per completare questa tesi, e per tutto ciò che mi hanno insegnato e che mi insegneranno.

Un ringraziamento particolare va alla mia famiglia, a Chiara e ai miei amici che mi hanno incoraggiato e che hanno sempre creduto in me.

INDICE

CAPITOLO 1

INTRODUZIONE

Lo scopo di questa tesi è la realizzazione di un meta-modello di obblighi e divieti adatto a definirli e a determinarne lo stato, in base all'evoluzione temporale di un sistema nel quale si rappresentano eventi, azioni ed, in generale, interazioni tra agenti, che possono essere soggetti a vincoli economici o legali. Questo tipo di modello può essere applicato a sistemi multiagente, a generiche interazioni tra entità su Internet, a rapporti di natura commerciale e a molto altro ancora.

Dato che l'obiettivo di tale modello è quello di essere ampiamente riutilizzabile in molteplici applicazioni anche differenti, un prerequisito fondamentale è l'interoperabilità dei dati descritti attraverso questo modello. Per ottenere questa proprietà abbiamo scelto di utilizzare il linguaggio OWL raccomandato dal World Wide Web Consortium (W3C).

La particolarità del modello proposto è la descrizione di eventi grazie alla quale è possibile ottenere una struttura molto flessibile di obblighi e divieti che si riferiscono a tali eventi, e il modo in cui questi sono intercambiabili nell'intervenire nel ciclo di vita dei vincoli rappresentati.

La tesi è organizzata nel modo seguente: nel prossimo capitolo analizzeremo l'obiettivo della tesi con riferimento ad altri modelli esistenti per la rappresentazione di obblighi e divieti e i relativi limiti, oltre a esporre gli strumenti utilizzati per lo sviluppo del progetto. Nel capitolo 3 descriveremo cosa si intende per obbligo e divieto nel linguaggio comune, quali tipi di obbligo differenti si possono rappresentare, i concetti correlati di agente ed evento e il ciclo di vita di questi vincoli. Nel capitolo 4 presenteremo gli aspetti più importanti dell'ontologia sviluppata e dell'applicazione Java che gestisce gli eventi temporali e gli stati degli obblighi. Mostreremo poi un esempio di utilizzo dell'ontologia nel capitolo 5, rappresentando obblighi che derivano da una transazione economica e i possibili sviluppi degli eventi. Nel capitolo 6 riassumeremo lo scopo del progetto, quanto realizzato e i possibili sviluppi del progetto stesso. Infine, nell'appendice A, presenteremo nel dettaglio le classi e le proprietà definite nell'ontologia, e, nell'appendice B, due esempi di ontologie di dominio che descrivono tipi reali di eventi, con l'intento di mostrare il processo di definizione di ontologie basate sui concetti di relazioni deontiche in un preciso ambito di interesse.

CAPITOLO 2

SPECIFICHE DEL PROGETTO

2.1. Scopo del progetto

La definizione delle interazioni tra agenti eterogenei è un aspetto importante nello sviluppo di molteplici applicazioni e sistemi distribuiti, soprattutto quando si opera in ambiente Web, ad esempio nel mondo dell'e-commerce. In queste situazioni si ha spesso bisogno di regolare le suddette interazioni tramite vincoli di natura legale, tipicamente descrivendo le azioni che un agente può compiere o, viceversa, che non deve effettuare, in base al verificarsi di svariate condizioni o durante determinati periodi temporali. Questo genere di vincoli comporta la necessità di regolare le azioni di un agente oppure determinare le violazioni che portano a conseguenze economiche o legali. Perciò è fondamentale definire tali concetti in modo da avere uno strumento per verificare lo stato di obblighi e divieti di un agente.

L'obiettivo di questo progetto è l'implementazione di un sistema basato su Ontologie Web per *definire*, *gestire*, e *monitorare* obblighi e divieti, con lo scopo di formalizzare concetti quali contratti, accordi, promesse con valore legale e interazioni tra partner commerciali.

Le ontologie web sono una parte di ciò che è chiamato Semantic Web [1][2] o Web 3.0 [3] che ha lo scopo di rendere facilmente accessibili i dati su internet: per ottenere questo risultato è necessario rappresentare le informazioni presenti nei documenti reperibili nel web in modo comprensibile e manipolabile tramite procedure automatiche. In altre parole, è necessario formalizzare la semantica dei dati che si digitalizzano, in modo da rendere i computer in grado di comprenderne il significato, anziché trattarli come semplici sequenze di bit. I vantaggi nell'utilizzo di ontologie derivano dalla possibilità di sfruttare procedure di calcolo standard per la manipolazione dei dati: questi strumenti sono ben consolidati e ottimizzati per ottenere prestazioni elevate e proprietà importanti come la decidibilità. Inoltre la rappresentazione attraverso logiche descrittive permette un alto livello di interoperabilità tra applicazioni in ambito web, grazie alla diffusione del web semantico.

Il successo del web semantico ha fatto emergere la necessità di definire modelli con la possibilità di essere riutilizzati ogni qualvolta vi sia l'esigenza di rappresentare i medesimi concetti. Anche in questo caso, una delle caratteristiche fondamentali per la creazione del modello è la generalità. I concetti di obblighi e divieti sono molto comuni nelle relazioni tra agenti e nelle transazioni che quotidianamente avvengono in Internet.

La definizione di un meta-modello utile, non solo per rappresentare gli obblighi, ma anche per analizzare e monitorare lo stato di questi obblighi, è dunque un progetto ambizioso, poiché necessita la definizione formale di concetti astratti, con la possibilità di essere applicati a qualsiasi ambito, ma anche la determinazione del comportamento di individui concreti che rappresentano gli obblighi. Pertanto, è necessario che il ciclo di vita di un obbligo sia indipendente dal tipo degli eventi a cui si riferisce.

Nella descrizione degli eventi bisogna inoltre considerare che possono esserci diverse rappresentazioni per lo stesso evento, in base al livello di dettaglio che si sceglie di esprimere. È normale dunque che un obbligo sia riferito ad un tipo di azione descritto in modo generico e che qualsiasi azione concreta, che ne rispecchia la struttura, può influenzarne lo stato. Ad esempio, l'obbligo di pagare le tasse descrive l'importo della tassa da pagare ad un determinato ente, ma tipicamente vi sono varie modalità per effettuare il pagamento: bollettini postali, addebito sul conto, carte di credito, eccetera. Tutte queste modalità soddisfano l'obbligo poiché sono rappresentazioni più specifiche della medesima azione.

2.2. Altri modelli di obblighi e divieti

Esistono vari modelli per la descrizione di obblighi e divieti definiti con molteplici strumenti e tipi di rappresentazioni differenti. Prendiamo in considerazione un modello basato su ontologie OWL precedentemente definito dal Professor Marco Colombetti[1,2] e dalla Professoressa Nicoletta Fornara[1][4].

Questo modello propone di rappresentare obblighi e divieti come proposizioni temporali (rispettivamente positive e negative) che descrivono il vincolo rispetto ad un'azione in un determinato istante di tempo. Inoltre descrive un "impegno" (Commitment) come una proposizione temporale che rappresenta il vincolo vero e proprio, una seconda proposizione temporale che indica la condizione di validità dell'impegno e un agente che contrae questo impegno nei confronti di un secondo agente.

Tale modello ha alcuni limiti che ci proponiamo di risolvere nel seguito del testo: un primo difetto è che non vi è differenza tra la descrizione di un'azione che influisce sullo stato di un vincolo e la realizzazione concreta dell'azione stessa. Ciò può creare confusione quando si vuole stabilire quali eventi o azioni sono stati effettuati, e quali invece sono solo rappresentazioni del contenuto di un obbligo o divieto. Questo inoltre non permette di considerare la realizzazione di obblighi attraverso azioni equivalenti o sussunte. Un secondo limite è la validità delle proposizioni temporali, che sono fissate a precisi istanti di tempo, senza poter esprimere vincoli dipendenti da generiche azioni dove non è noto a priori l'istante in cui avvengono. Si pensi, ad esempio, a obblighi derivanti da azioni di altri agenti o eventi atmosferici o altri tipi ancora. Infine, gli stati degli impegni sono rappresentati come sottoclassi di Commitment, senza associare il periodo di tempo in cui un impegno assume tali stati. Da ciò deriva che un impegno può risultare pendente e contemporaneamente soddisfatto oppure violato.

1 Università della Svizzera italiana, via G. Bu± 13, 6900 Lugano, Switzerland
fnicoletta.fornara@lu.unisi.ch, marco.colombettig@lu.unisi.ch,
2 Politecnico di Milano, piazza Leonardo Da Vinci 32, Milano, Italy
marco.colombetti@polimi.it

2.3. Descrizione tecnologie utilizzate

2.3.1.OWL

Il linguaggio OWL 2 [5] è un'estensione di OWL basato sulla logica descrittiva *SROIQ*(D*n*)ed è raccomandato dal W3C dal 2009. La logica SROIQ(D*n*) è un sottoinsieme decidibile della logica del prim'ordine, scelta per avere un grado di espressività adeguato, mantenendo la decidibilità del ragionamento deduttivo applicabile ai concetti che si possono esprimere.

In OWL è possibile definire concetti come *classi*, *proprietà* ed *attributi* ed anche le loro caratteristiche. Un'ontologia OWL contiene tre parti: la Terminological Box (TBOX), che contiene le classi e le relazioni tra queste, la Role Box (RBOX), che descrive le proprietà e gli attributi, e l'Assertion Box (ABOX) che contiene gli individui noti dell'universo, le classi a cui appartengono, e le proprietà ed attributi che possiedono.

Le classi sono la rappresentazione di sottoinsiemi dell'universo, che raccolgono individui con determinate caratteristiche in comune. Tra classi diverse ci possono essere molteplici relazioni, come l'equivalenza o la proprietà di sottoclasse, ed è possibile costruire nuove classi con gli operatori classici della logica degli insiemi (intersezione ed unione).

Le proprietà descrivono relazioni tra due individui dell'universo. La definizione di proprietà comprende dominio e codominio e altre caratteristiche come transitività, simmetria e riflessività. Poiché le proprietà rappresentano insiemi di coppie dell'universo, è possibile definire anche assiomi di sottoproprietà, i quali indicano che ogni coppia appartenente ad una proprietà R appartiene anche ad una proprietà P. Gli attributi sono relazioni tra individui e dati, come ad esempio valori numerici, date o stringhe (tipi di dato XML).

Gli individui infine rappresentano oggetti dell'universo. È possibile asserire l'uguaglianza o la diversità di due individui, definire l'appartenenza di un individuo ad una specifica classe, affermare che possiede alcuni attributi o che è correlato ad un altro individuo da una certa proprietà.

Un aspetto che distingue le basi di conoscenze dalle base di dati è la possibilità di condurre *ragionamenti* di tipo deduttivo in modo automatico. Un ragionamento è dunque un procedimento che porta a verificare se un enunciato o asserzione *X* (ad esempio una relazione di sottoclasse) è *conseguenza logica* di una base di conoscenze.

Un reasoner, vale a dire un sistema software che implementa servizi di ragionamento, è in grado, dato un insieme di assiomi di TBOX ed RBOX e un insieme di asserzioni di ABOX, di dedurre nuove informazioni che sono conseguenza logica di quanto presente nell'ontologia.

2.3.2.SWRL

Alcuni reasoner possono interpretare il linguaggio SWRL (Semantic Web Rule Language) [6] con il quale è possibile definire un insieme di regole (detto Rule Boxe) con una sintassi simile a quella di Datalog [7]. Una regola è composta da una parte sinistra, l'antecedente, e una parte destra, il conseguente, separate dal simbolo di implicazione. L'antecedente contiene uno o più predicati unari o binari in congiunzione: quando tutti questi predicati sono verificati allora è possibile dedurre il predicato alla destra del simbolo " → ".

I predicati unari rappresentano l'appartenenza di un individuo, o di una variabile, ad una classe, mentre i predicati binari rappresentano la proprietà tra due individui o variabili, oppure l'attributo di un individuo: MyClass(?var), MyProperty(myIndiv,?var), MyAttribute(?var,data), MyAttribute(myIndiv,?var).

Le variabili, quantificate universalmente, possono rappresentare individui noti dell'universo oppure dati XML. Non è possibile invece denotare oggetti dell'universo non rappresentati da individui, poiché altrimenti il ragionamento che ne deriverebbe sarebbe indecidibile.

Oltre ai predicati è possibile utilizzare nella parte sinistra delle regole SWRL alcuni built-in, simili nella forma a normali predicati, ma che permettono di effettuare operazioni sui dati: per esempio, è possibile confrontare date oppure eseguire operazioni su valori numerici.

Un reasoner in grado di interpretare le regole SWRL può quindi usarle non solo per dedurre il conseguente, ma anche per dedurre la negazione dell'antecedente se il conseguente è negato.

2.3.3.SPARQL-DL

SPARQL-DL [8] è un sottoinsieme del linguaggio di query SPARQL (SPARQL Protocol And RDF Query Language) ideato per ottenere informazioni da dati in formato RDF [9] e raccomandato dal W3C per interrogare ontologie OWL.

Il linguaggio SPARQL definisce una sintassi simile ad SQL per recuperare dati da insiemi di triple "soggetto-predicato-oggetto" tipiche della rappresentazione RDF. SPARQL interpreta tutti i dati in formato RDF e non solo OWL, perciò non è possibile ottenere risultati attraverso ragionamento deduttivo. SPARQL-DL, invece, può essere utilizzato abbinandolo a servizi di ragionamento ed è quindi possibile ottenere non solo risultati espressamente riportati nell'ontologia interrogata, ma anche tutti i dati che è possibile dedurre dall'insieme di triple. SPARQL-DL permette di combinare facilmente query di TBOX, RBOS ed ABOX, rendendolo dunque un linguaggio molto espressivo e particolarmente adatto all'utilizzo con ontologie OWL.

Con SPARQL-DL, inoltre, è possibile applicare la Negation As Failure, ovvero determinare l'insieme di individui che non possiedono una caratteristica come differenza insiemistica degli individui noti dell'universo eccetto quelli che possiedono tale caratteristica. Quest'operazione non è possibile tramite normali servizi di ragionamento, poiché la logica descrittiva SROIQ(D*n*) si basa sull'Open World Assumption.

Il reasoner Pellet, è in grado di interrogare un'ontologia con query SPARQL-DL, combinandole ai propri servizi di ragionamento, oltre a interpretare le regole SWRL per estendere le ontologie OWL.

2.3.4. 4D-FLUENTS

Il modello 4D-Fluents [10] propone una soluzione alla rappresentazione di informazioni variabili nel tempo in ontologie OWL. Esistono vari modi in OWL per rappresentare informazioni temporali: il più noto ed utilizzato è la reificazione delle proprietà. Una proprietà tra due individui valida in un determinato periodo temporale è sostituita da un terzo individuo, che unisce i primi due, oltre alla rappresentazione dell'intervallo di tempo in cui ha validità. Questo metodo, sebbene efficace, introduce alcuni problemi nell'ontologia tra cui la proliferazione degli individui e la difficoltà di evitare la duplicazione delle relazioni, ovvero più individui che identificano la stessa relazione nello stesso periodo temporale. Il difetto più importante di questo approccio, tuttavia, è che, rappresentando una relazione come un oggetto, si perde la possibilità di descrivere delle caratteristiche importanti della proprietà stessa come ad esempio la transitività, la riflessività, la simmetria, e la possibilità di utilizzare la proprietà nella definizione di altre classi o proprietà OWL grazie a restrizioni di cardinalità o proprietà inverse. Questo è il motivo per cui il modello 4D-Fluents propone un modo alternativo di rappresentare informazioni temporali, con la possibilità di mantenere l'espressività delle proprietà OWL.

Il modello definisce come perdurant tutte le entità che non cambiano nel tempo e endurant quelle entità che sono valide per un certo periodo di tempo. Per rappresentare proprietà che variano nel tempo, si possono dunque creare individui, detti parti temporali, che si riferiscono alle entità perdurant, correlate ad un intervallo nel quale le rappresentano. In questo modo se due individui possiedono una certa proprietà, questa relazione sarà rappresentata da una proprietà che unisce le parti temporali dei due individui, anziché loro stessi.

Ad esempio si possono definire una società e una persona come entità perdurant e poi definire le parti temporali (endurant) che le rappresentano solo per un intervallo di tempo T. Per descrivere la relazione di “Amministratore Delegato” tra le due entità durante T, è sufficiente associare tramite la proprietà *CEOof* le parti temporali dell'azienda e della persona.

Questo approccio implica l'introduzione di nuovi individui nell'ontologia, come nel caso della reificazione, ma ha il grande vantaggio di poter definire le “proprietà temporali”, dette fluents, come transitive e simmetriche, oltre a poter rappresentare la proprietà inversa, restrizioni di cardinalità e tutte le operazioni sulle proprietà che fanno parte dell'espressività di OWL.

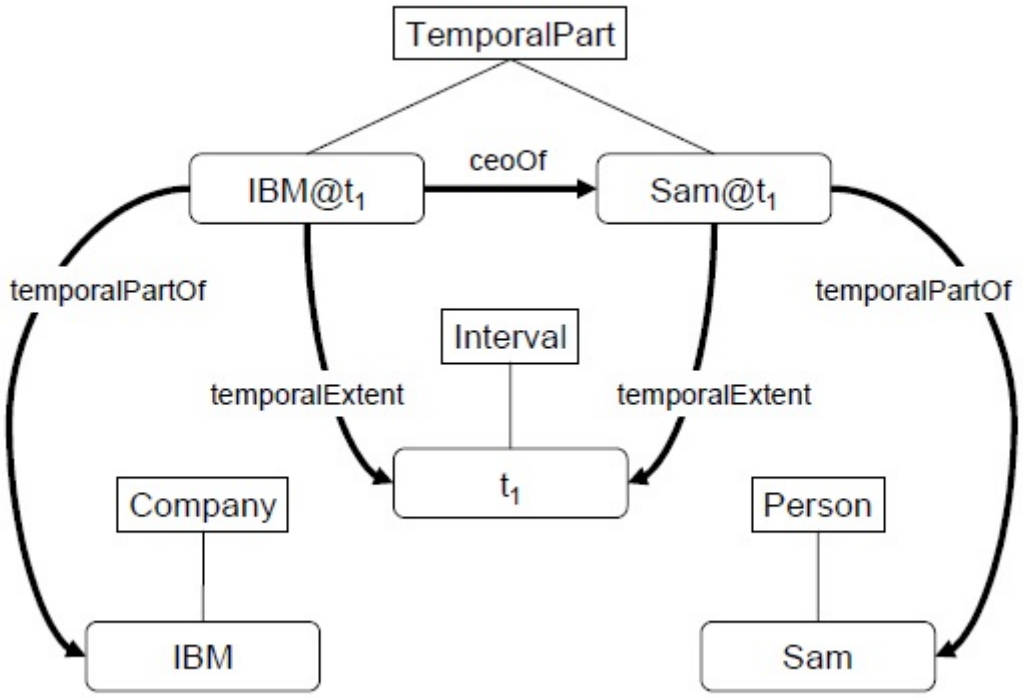

Figura 2.1 – 4D-Fluents

CAPITOLO 3

MODELLO DELLE RELAZIONI DEONTICHE

In questo capitolo analizzeremo la struttura del concetto di obbligo e di divieto dal punto di vista astratto. La prima sezione riguarda l'analisi del concetto di obbligo e di divieto, dei tipi di obbligo e le loro principali caratteristiche, i punti in comune e le differenze. La seconda affronta la correlazione tra i concetti di obbligo e di divieto e come possono essere paragonati. La terza sezione contiene un'analisi dei concetti correlati agli obblighi e la loro struttura. La quarta sezione descrive le fasi del ciclo di vita di un obbligo.

3.1. Le Relazioni Deontiche

Un *obbligo* o *dovere* è un'imposizione a compiere una o più azioni, da parte di un soggetto, tipicamente a beneficio di un secondo soggetto, a seguito di un vincolo morale o legale. Un obbligo può riferirsi ad azioni *positive* oppure ad azioni *negative* e, in questo caso, è denominato *divieto* o *proibizione*. Per semplicità, in questo testo, parleremo di obbligo quando si tratta di azioni positive e di divieto quando si tratta di azioni negative. Per indicare poi un generico vincolo, includendo sia obblighi sia divieti, utilizzeremo il termine *relazione deontica*.

Analizzando il concetto di obbligo e il suo utilizzo nel linguaggio comune si possono determinare quattro tipi distinti di obbligo (e analogamente quattro tipi di divieto):

- Obbligo singolo o puntuale
- Obbligo periodico o ripetuto
- Obbligo complesso
- Obbligo continuo

Il caso più semplice è l'obbligo singolo, vale a dire quel tipo di obbligo che implica lo svolgimento di un'azione una singola volta per essere considerato soddisfatto. Questo è il caso più comune di obbligo.

Il caso di obbligo ripetuto comprende quelle situazioni in cui è necessario compiere la medesima azione ripetutamente, durante un certo periodo di tempo, o ad intervalli regolari. Questo caso è in realtà assimilabile all'aggregazione di tanti obblighi singoli quanto è il numero di volte in cui l'azione deve essere ripetuta.

Per obbligo complesso si intende indicare una situazione in cui per soddisfare l'obbligo è necessario compiere più azioni di tipo differente. Come per il precedente, anche questo tipo di obbligo è in realtà determinato dal soddisfacimento di più obblighi semplici, uno per ogni azione da compiere.

Infine, per obbligo continuo si identifica un caso particolare e più raro di obbligo in cui una certa azione deve durare per tutto un periodo di tempo o al verificarsi di determinate condizioni. Va però sottolineato che è facile confondere questo tipo di obbligo con ciò che in realtà può essere rappresentato con un divieto. Ad esempio: l'obbligo a mantenere il silenzio durante la visione di un film in una sala cinematografica è equivalente al divieto di parlare o genericamente emettere rumori. Questo esempio ci permette di introdurre la proprietà della dualità, che mette in comune obblighi e divieti.

Nel resto del testo prenderemo in considerazione gli obblighi come obblighi singoli.

Figura 3.1 – Tipi di Obbligo.

3.2. La dualità tra obbligo e divieto.

Un obbligo è soddisfatto al verificarsi, almeno una volta, dell'azione considerata a partire dal momento in cui l'obbligo diventa attivo o valido. Se l'azione non si verifica prima di un certo evento o istante di tempo, l'obbligo è violato.

Per i divieti ciò che accade è esattamente l'opposto: un divieto è violato se si verifica l'azione almeno una volta, oppure è soddisfatto se non si verifica entro la condizione di termine del divieto.

Analizzando la relazione tra questi concetti, si nota una certa somiglianza: è quindi possibile descrivere un divieto come un obbligo negativo? In realtà i due concetti sono correlati, ma non identici. In base ai tipi di obbligo che abbiamo introdotto precedentemente, un divieto non è esattamente identico all'obbligo (semplice) di non compiere l'azione, poiché, in questi termini, basterebbe non compiere l'azione per un istante per considerare l'obbligo soddisfatto.

Questo, come si può intuire, non è lo stesso risultato che si vuole ottenere con un divieto. Ad esempio, il divieto di fumare è violato dall'azione di accendersi una sigaretta, mentre l'obbligo di non fumare sarebbe soddisfatto se almeno per un istante il soggetto non stesse fumando.

$$Obbligo(\neg A) \neq Divieto(A)$$

3.3. Gli agenti e gli eventi

Un obbligo (e analogamente un divieto) è descritto dagli agenti che coinvolge e da uno o più eventi che ne modificano lo stato.

Per agente si intende una generica entità che può svolgere azioni che possono influire sullo stato degli obblighi e divieti. Esempi di agente sono le persone fisiche, le società, gli enti, ma anche gli animali o addirittura agenti artificiali (si pensi alla possibilità di utilizzare procedure automatiche per partecipare ad aste online su Ebay).

Un generico evento è un avvenimento che si verifica in un determinato istante e può avere varie caratteristiche. Ci sono molteplici tipi di evento quali: eventi temporali, eventi atmosferici o eventi compiuti da un agente, più comunemente detti “azioni”, come lanciare una moneta, leggere un libro, acquistare un prodotto, eccetera.

Un obbligo si riferisce ad un'azione: l'obbligo è detto soddisfatto quando l'azione di riferimento è compiuta dall'agente che ha contratto l'obbligo. Spesso è utile affermare che un obbligo sia valido entro un determinato periodo di tempo, o fino a quando si verifichi un preciso evento. Per esempio, Anna ha l'obbligo di ordinare la propria stanza entro le 7 di sera oppure prima che i suoi genitori tornino dal lavoro. In questo caso l'obbligo si riferisce all'azione di mettere ordine ed è valido fino al verificarsi del ritorno a casa dei genitori. Se Anna svolge questa azione, ha soddisfatto l'obbligo, altrimenti lo ha violato.

Un obbligo, inoltre può essere valido solo da un certo istante in poi o qualora accada un determinato evento. Per esempio, se i genitori di Anna tornano tardi dal lavoro, lei ha l'obbligo di preparare la cena.

Un tipo particolare di evento è quello che influisce direttamente su un obbligo, generandolo o cancellandolo. Ad esempio una promessa è un'azione: Anna promette a sua madre di finire i compiti dopo cena. Questo però genera anche l'obbligo di Anna di finire i compiti, a partire dal momento in cui finiscono di cenare e prima di andare a letto.

3.4. Stati delle relazioni deontiche

Come accennato nel paragrafo precedente, un obbligo evolve attraverso vari stati dal momento in cui viene creato. Inizialmente l'obbligo è inattivo: può quindi essere cancellato oppure diventare attivo nel momento in cui si verifichi un determinato evento di attivazione. Quando attivo, l'obbligo può mutare in altri 2 stati: se si verifica l'azione imposta dall'obbligo, questo assume lo stato di obbligo soddisfatto, mentre se si verifica la condizione di terminazione (e l'azione non è stata svolta) l'obbligo diventa violato. Un obbligo può essere inoltre cancellato anche se già attivato, ma non se è già stato soddisfatto o violato.

Soddisfatto e violato sono stati finali, poiché una volta in cui l'obbligo assume uno di questi stati non può più variare. Infatti, soddisfatto un obbligo non è ovviamente possibile che questo venga violato e viceversa.

Meno banale è invece lo stato cancellato. In base al ciclo di vita di un obbligo possono seguire determinati eventi, ad esempio una sanzione in caso di violazione o, viceversa, un premio in caso di soddisfacimento dell'obbligo. Per questo motivo sarebbe sbagliato pensare che un obbligo possa essere cancellato se è già stato soddisfatto oppure violato.

Il ciclo di vita di un obbligo è rappresentata graficamente nella Figura 3.3, dove i cerchi rappresentano lo stato assunto dall'obbligo, abbreviati in NA, A, C, S e V, e le frecce rappresentano le transizioni di stato con relativa descrizione. Gli stati finali sono rappresentati con un cerchio doppio.

In maniera duale si definisce il ciclo di vita di un divieto, come rappresentato nella Figura 3.4.

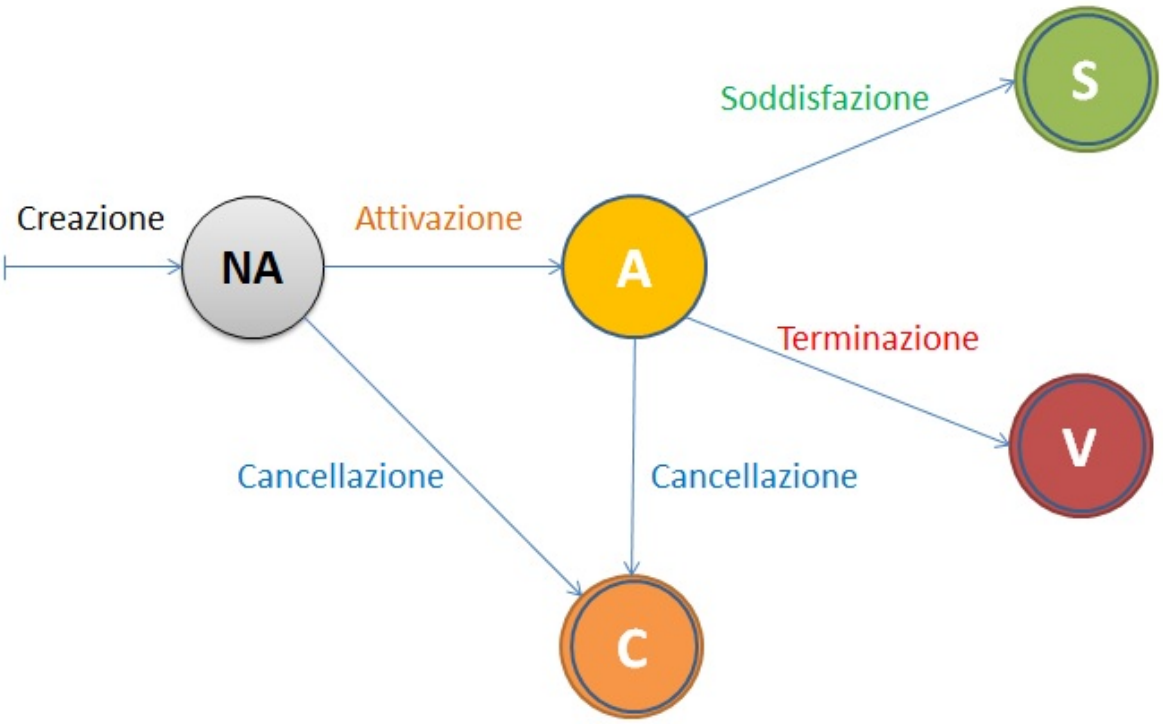

Figura 3.3 – Ciclo di vita di un Obbligo

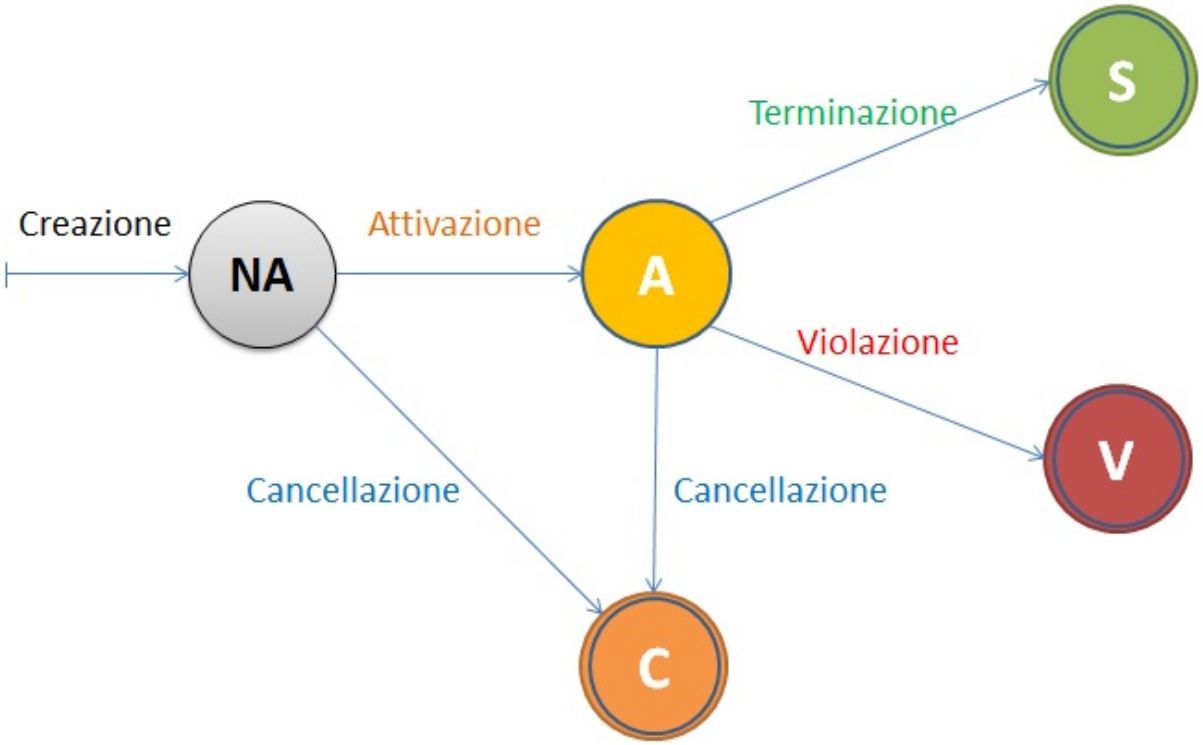

Figura 3.4 – Ciclo di vita di un Divieto

CAPITOLO 4

REALIZZAZIONE IN OWL

In questo capitolo analizzeremo la struttura dell'ontologia realizzata in OWL. La prima parte tratterà della struttura scelta per rendere componibile a piacimento l'ontologia. Successivamente affronteremo la descrizione degli eventi attraverso la definizione di 3 livelli concettuali. Proseguiremo descrivendo la tecnica per tenere traccia del ciclo di vita di una relazione deontica e degli stati che assume. Nella quarta sezione esamineremo come gli eventi interagiscono con gli obblighi e i divieti. In seguito tratteremo un limite rappresentativo del linguaggio OWL e come ricade sul progetto. Infine, nell'ultima parte, giustificheremo l'introduzione di Java, per quali funzionalità e con quali accorgimenti.

4.1. Struttura a tempio greco

Uno degli obiettivi di questo progetto e dell'ontologia realizzata è quello di renderla idonea ad adattarsi a qualunque azione ed evento che si possono rappresentare e sui quali si vogliono definire relazioni deontiche. Per rendere utilizzabile questa ontologia, che chiameremo top ontology, in ambiti o domini diversi, i concetti definiti devono necessariamente essere generici e astratti. È necessario definire successivamente gli eventi che si utilizzano in una specifica applicazione e le proprietà degli stessi.

Queste ontologie che descrivono tipi concreti di evento, dette ontologie di dominio, possono essere definite in modo da essere sfruttate in varie applicazioni, senza effettuare modifiche e, tipicamente, utilizzandone più alla volta. Infine, un'ontologia potrebbe importare la top ontology e varie ontologie di dominio per rappresentare obblighi e divieti riferiti a un particolare campo di interesse.

Volendo raffigurare con un diagramma questa struttura, si ottiene, con un pizzico di fantasia, uno schema somigliante ad un tempio greco (Figura 4.1):

Figura 4.1 – Tempio Greco.

Questa struttura permette la definizione di molteplici ontologie di dominio, indipendenti ed esaustive rispetto a un certo tipo di eventi, che possono essere riutilizzate in diversi campi di interesse. Alla base del tempio si realizzano ontologie applicative che comprendono solamente dati, rappresentati in OWL secondo le classi e proprietà definite dalle ontologie ai livelli superiori, creando così una netta separazione tra il modello e i dati.

4.2. Gli eventi

Un obbligo è riferito ad alcuni eventi che ne influenzano lo stato. Per rappresentare un evento correlato ad un obbligo è necessario fare una precisazione: ciò a cui si riferisce l'obbligo è la descrizione di un evento; quando l'evento diventa concreto cambia effettivamente lo stato dell'obbligo. Un evento concreto, oltre ad una descrizione, riporta l'istante a cui avviene e, nel caso di azioni, l'agente che lo compie. Questa separazione è in realtà alquanto comune quando si utilizzano Eventi in OWL (Figura 4.2).

La descrizione è un punto delicato nella definizione degli eventi, poiché è importante considerare che eventi di tipo *diverso*, ma *simile* possono ugualmente attivare, soddisfare o violare un obbligo o un divieto. Per esempio, volendo definire l'obbligo di pagare una certa somma di denaro, dobbiamo considerare che esistono varie forme di pagamento (contanti, carta di credito, bancomat) e che, a seconda dei casi, tutte o alcune possono soddisfare l'obbligo. In altre parole è necessario definire una gerarchia di eventi in modo tale da poter sostituire un tipo di evento ad uno di tipo superiore, ovvero più generico. In questo senso un "Pagamento" è un tipo di evento più generico di un "Trasferimento bancario" o di un "pagamento in contanti".

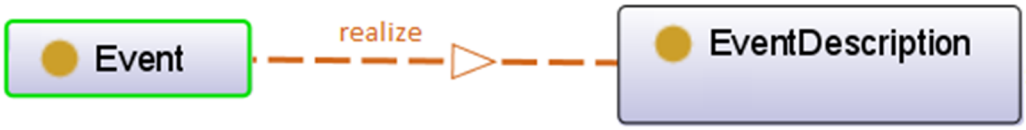

Figura 4.2 – Eventi Concreti e Descrizioni di eventi.

Verrebbe dunque spontaneo rappresentare i vari tipi con una gerarchia di classi e sottoclassi in OWL. Questo però rende difficile controllare che un evento sia effettivamente sostituibile ad un altro. Le descrizioni di evento tipicamente sono più dettagliate di un semplice tipo di evento, ma contengono anche proprietà ed attributi. Nel caso dei pagamenti vanno considerati anche l'*ammontare di denaro* e gli *agenti* coinvolti (colui che versa e colui che riceve la somma di denaro).

Come fare per associare due pagamenti di tipo diverso ma con gli stessi "parametri"? Una possibilità è quella di definire questa proprietà con regole SWRL, che verificano se un individuo appartiene a una sottoclasse di pagamento, con il medesimo ammontare e con gli stessi agenti coinvolti. Questo però implicherebbe l'introduzione di una regola SWRL per ogni coppia di classi all'interno della gerarchia dei Pagamenti e lo stesso per la gerarchia delle Consegne e per ogni ontologia di dominio considerata.

Una soluzione per evitare la definizione di tutte queste regole in modo manuale è quella di *reificare* i tipi di evento, ovvero introdurre degli individui nell'ontologia che rappresentano i concetti astratti di tipo di evento, affiancandoli alle descrizioni di eventi e agli eventi concreti (Figura 4.3).

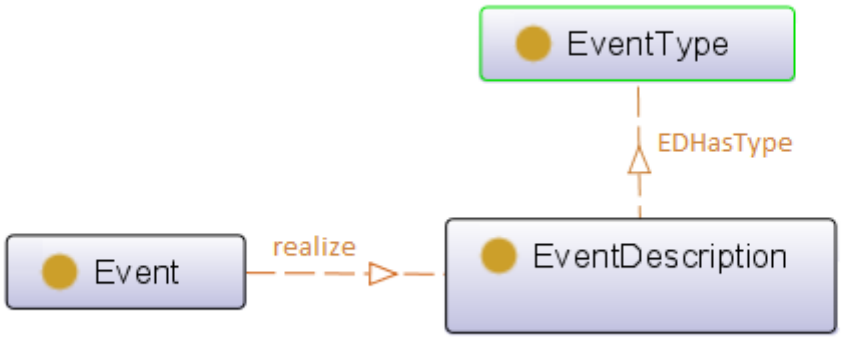

Figura 4.3 – Eventi Concreti, Descrizioni di eventi e Tipi di eventi.

Con la reificazione si associa una descrizione di evento a un individuo che ne identifica il tipo, tramite la proprietà OWL HasType. La gerarchia dei tipi si crea tra gli individui con un'altra proprietà OWL (subTypeOfEvent), anziché tramite assiomi di sottoclasse. Per stabilire se un evento A è sostituibile ad un altro evento B è possibile utilizzare un'unica regola SWRL, che controlla l'equivalenza dei "parametri" degli eventi e che tra i tipi dei due eventi vi sia relazione di *sottotipo (Figura 4.4).*

Questo permette di definire una sola regola per ogni gerarchia di tipi, in quanto i "parametri" degli eventi non cambiano all'interno di una singola gerarchia, ma generalmente variano tra gerarchie diverse.

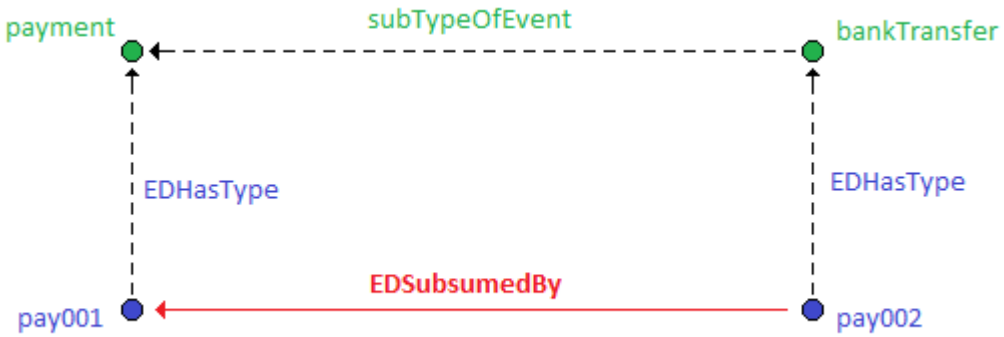

Figura 4.4 – Sussunzione di eventi

4.3. Gli stati delle relazioni deontiche

Abbiamo già presentato nel capitolo 2 il modello *4D Fluents* per la rappresentazione dell'evoluzione temporale dell'informazione all'interno di un'ontologia. Questo modello è stato utile per definire lo stato delle relazioni deontiche, in quanto si tratta di un'informazione che varia nel tempo e di cui è necessario tener traccia.

In seguito a queste considerazioni abbiamo scelto di rappresentare gli stati di un obbligo come individui separati dagli obblighi stessi, appartenenti quindi a una classe disgiunta, anziché come una sottoclasse degli obblighi. Tale classe descrive le parti temporali di relazioni deontiche: *DRTemporalPart*. Quindi lo stato di un obbligo è definito non più da una sottoclasse di *DeonticRelation,* come si pensava all'inizio, ma come un individuo separato, associato all'obbligo o al divieto tramite la proprietà *temporalPartOf* e all'intervallo di tempo in cui assume quel determinato stato tramite la proprietà *temporalExtent,* che li associa ad un individuo di classe *Interval (Figura 4.5)*.

Questo permette di rendere monotòna la rappresentazione degli stati, poiché il cambio di stato di una relazione deontica si traduce nella creazione di un nuovo individuo, appartenente alla classe delle parti temporali, associato alla relazione deontica e all'intervallo di validità. Gli stati precedentemente assunti rimangono salvati e associati ad intervalli conclusi.

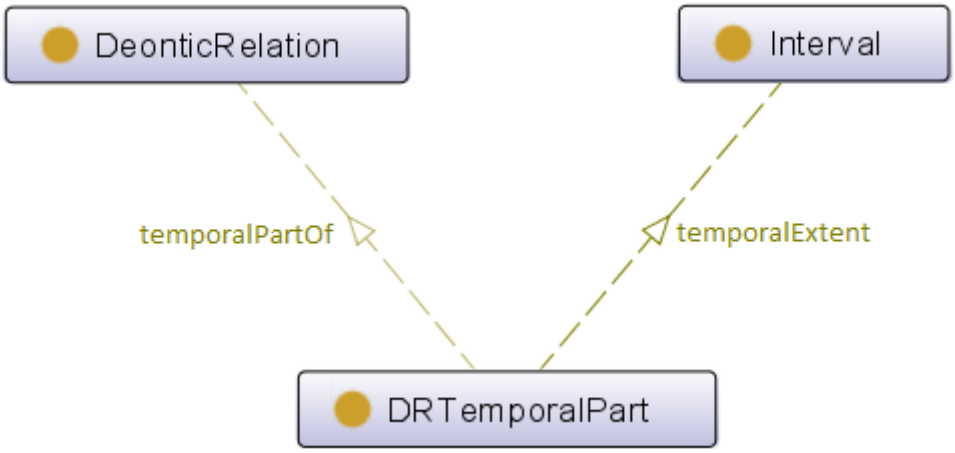

Figura 4.5 – Temporal Part.

Per determinare lo stato in cui si trova un obbligo è sufficiente selezionare l'ultima parte temporale riferita a quell'obbligo e controllare a quale delle sottoclassi disgiunte di *DRTemporalPart* appartiene: *Activated*, *Canceled*, *Fulfilled*, *Violated* *(Figura 4.6)*.

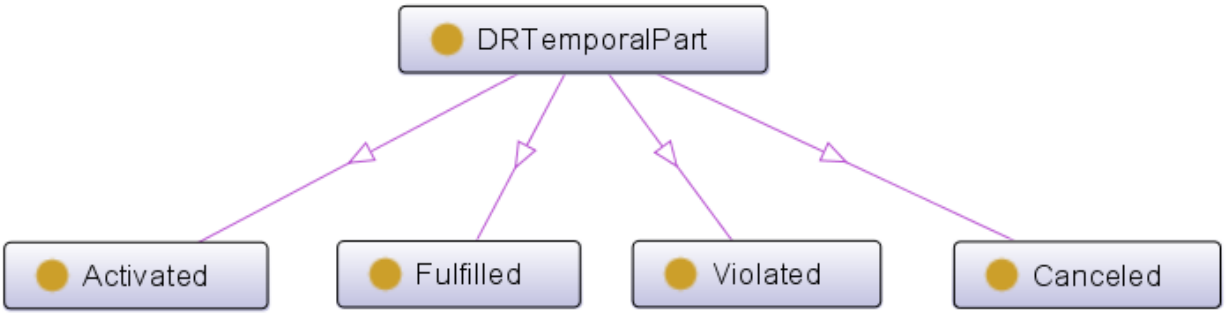

Figura 4.6 – Temporal Parts: Stati.

4.4. Come gli eventi influiscono sullo stato di una relazione deontica

Abbiamo visto come stabilire se un evento è più generico di un altro o, viceversa, se un evento è sussunto da un altro. Questo permette ad eventi sussunti di modificare lo stato di relazioni deontiche riferite a descrizioni di eventi più generiche. Ma come si determina, per esempio, se un evento soddisfa un obbligo?

Per poter determinare se un evento "è idoneo" a soddisfare un preciso obbligo, abbiamo scritto una regola SWRL nella top ontology e, analogamente, per gli altri stati delle relazioni deontiche. Analizziamo ora la struttura di queste regole, prendendo come esempio proprio quella per la soddisfazione di un obbligo, riportata di seguito:

Obligation(?o),content(?o, ?oblDescr),hasDebtor(?o, ?a), Agent(?a),
Event(?e),happensAt(?e, ?if),hasActor(?e, ?a),realize(?e, ?evDescr),
Activated(?tp),temporalPartOf(?tp, ?o),createTemporalPart(?eg, ?tp),
happensAt(?eg, ?ia),hasTime(?ia, ?ta), hasTime(?if, ?tf), lessThan(?ta, ?tf),
EDSubsumedBy(?evDescr, ?oblDescr)
-> *fulfill(?e, ?o)*

La regola è composta da una parte a sinistra del simbolo di implicazione →, composta da predicati unari e binari in congiunzione, e da una parte destra composta da un solo predicato unario o binario. I predicati unari permettono di verificare se un individuo appartiene a una determinata classe, mentre i predicati binari permettono di controllare se due individui sono connessi da una certa proprietà, oppure se un individuo possiede un certo attributo.

Alcuni reasoner, sono in grado di interpretare queste regole per estendere l'espressività dell'ontologia, sostituendo alle variabili (quelle precedute da punto interrogativo) gli individui noti dell'universo. Questa regola è alquanto complicata: per analizzarla la scomponiamo in alcune parti, evidenziandole con colori differenti.

La prima parte, corrispondente all'obbligo, comprende i seguenti predicati:

Obligation(?o),content(?o, ?oblDescr),hasDebtor(?o, ?a), Agent(?a),

La variabile “?o” contiene un individuo di classe Obligation. Essa ha una relazione con l'individuo “?oblDescr”, di classe EventDescription, che rappresenta il contenuto dell'obbligo (cioè l'azione che lo soddisfa), e con l'individuo “?a”, di classe Agent, che indica colui che contrae l'obbligo.

La seconda parte riguarda l'evento:

Event(?e),happensAt(?e, ?if),hasActor(?e, ?a),realize(?e, ?evDescr),

L'individuo “?e” di classe Event, accade all'Istante “?if”, è effettuato dall'agente “?a” e realizza la descrizione “?evDescr”.

La terza parte riguarda lo stato dell'obbligo:

Activated(?tp),temporalPartOf(?tp, ?o),createTemporalPart(?eg, ?tp),
happensAt(?eg, ?ia),hasTime(?ia, ?ta), hasTime(?if, ?tf), lessThan(?ta, ?tf),

La parte temporale “?tp” dell'obbligo “?o” è di classe Activated ed è stata creata dell'evento “?eg” (che attiva l'obbligo). Questo evento deve verificarsi prima dell'evento “?e” che soddisfa l'obbligo, perciò è necessario confrontare gli istanti di tempo in cui accadono i due eventi.

Infine occorre controllare che la descrizione dell'obbligo sia uguale o più generica rispetto alla descrizione dell'evento che è stato compiuto. Grazie alla struttura degli eventi, questo è semplicemente rappresentato da:

EDSubsumedBy(?evDescr, ?oblDescr),

Quando tutte queste condizioni sono verificate, allora l'evento “?e” “è idoneo” a soddisfare l'obbligo “?o”:

fulfill(?e, ?o)

Il fatto che l'evento “?e” sia *idoneo* non basta per stabilire che l'obbligo sia effettivamente soddisfatto. Infatti, se prima di questo evento se ne verificasse un altro che, con una regola analoga, termini l'obbligo, allora quest'ultimo non potrà più essere soddisfatto. Purtroppo non è possibile, all'interno della regola SWRL, verificare che l'obbligo non sia già terminato, quindi ci potrebbero essere più eventi con la proprietà *fulfill* rispetto allo stesso obbligo, ma al più uno di questi soddisfa l'obbligo, oppure nessuno qualora l'obbligo sia già terminato.

4.5. Negation As Failure

Per determinare se un obbligo sia soddisfatto o violato, è necessario verificare che sia avvenuto un evento sussunto dalla condizione corrispondente, ma in realtà questo non basta. Per poter asserire che un obbligo sia violato, oltre a controllare che la condizione di terminazione sia verificata, bisogna controllare che non siano avvenuti eventi precedenti che soddisfino l'obbligo.

Questo tipo di ragionamento è problematico in logiche descrittive come SROIQ, perché applicano l'assunzione di mondo aperto (OWA – Open World Assumption). Ciò significa che quanto è presente negli assiomi dell'ontologia è considerato vero, mentre tutto ciò che non è presente o deducibile, è indeterminato. In altre parole, l'informazione inserita nell'ontologia non è considerata completa, e questo permette di ottenere un ragionamento monotòno: l'aggiunta di nuove informazioni, se non in contraddizione con quelle esistenti, non cambia le informazioni che possono essere dedotte, ma è possibile dedurne di nuove.

In questo modo, però, un'interrogazione come "quali obblighi non sono stati soddisfatti?" non restituisce i risultati che ci si aspetta, poiché il reasoner restituisce solo quegli individui, per i quali la proprietà richiesta è espressamente indicata o dimostrabile.

Nel caso degli obblighi, abbiamo bisogno di determinare quando non si è verificato un evento che soddisfa l'obbligo, analogamente a quanto accade in una base di dati, per poter dire che questo è stato violato. La soluzione è quella di applicare la cosiddetta *Negation As Failure (NAF)* solo in particolari casi, ovvero quando è necessaria. La NAF permette, dato un predicato P(x), di determinare gli individui che non lo soddisfano (¬P(X)) come differenza insiemistica tra l'universo e l'insieme degli individui che hanno quella proprietà.

Ciò implica che l'informazione nella base di conoscenza sia completa. L'applicazione della NAF nell'ambito degli obblighi funziona solamente assumendo che, in un determinato istante, tutti gli eventi realmente accaduti siano stati rappresentati come individui all'interno dell'ontologia. Questa è un'assunzione molto forte, ma necessaria per determinare in un dato istante quali obblighi sono violati e, viceversa, quali divieti sono stati soddisfatti.

4.6. A cosa ci serve Java

La NAF, purtroppo, non è applicabile direttamente tramite i servizi offerti dal reasoner, ma possibile utilizzarla tramite alcuni linguaggi di query come ad esempio SPARQL-DL. È necessario dunque utilizzare applicazioni Java per interrogare con SPARQL-DL l'ontologia ed ottenere risultati corretti sotto l'assunzione di informazione completa.

Oltre alla necessità di interrogare l'ontologia, è stato necessario sviluppare una semplice applicazione Java per gestire l'inserimento di individui che rappresentano le parti temporali delle relazioni deontiche. Infatti, non è possibile tramite il reasoner immettere nuovi individui nell'ontologia, ma è necessario utilizzare le OWL API [11] per sviluppare del codice Java che funga da trigger, creando all'occorrenza le parti temporali degli obblighi e gli eventi temporali corrispondenti a istanti passati.

L'applicazione è stata dunque creata in modo tale da poter inserire dei dati nell'ontologia (agenti, eventi e relazioni deontiche) e monitorare l'evoluzione degli stati inserendo automaticamente le parti temporali.

L'ontologia e l'applicazione sono state sviluppate in modo tale da separare nettamente il ragionamento dalla gestione delle parti temporali. Infatti, come mostrato nei paragrafi precedenti, tramite le regole SWRL definite nell'ontologia, il reasoner è in grado di determinare se un evento sia idoneo a modificare in qualche modo lo stato di un obbligo e, grazie all'applicazione della NAF, possiamo verificare se il cambio di stato sia effettuabile, evitando, ad esempio, che un obbligo violato diventi soddisfatto e viceversa.

L'applicazione si limita quindi a interrogare l'ontologia tramite query SPARQL-DL che restituiscono coppie di eventi e relazioni deontiche, che vengono modificate da tali eventi. In base ai risultati ottenuti, creano nuovi individui che rappresentano lo stato attuale delle relazioni deontiche, associandoli ad esse, all'intervallo temporale di validità e all'evento che ha generato il nuovo stato.

CAPITOLO 5

USE CASE

In questo capitolo presenteremo un esempio di utilizzo dell'applicazione e vedremo gli individui inseriti in un'ontologia applicativa che importa i concetti della top ontology Obligation e delle due ontologie di dominio Payment e Delivery.

5.1. Descrizione dell'esempio

Un possibile ambito di utilizzo dell'ontologia è quello di monitorare gli obblighi che derivano dalla compravendita di un prodotto online.

Un agente, che chiameremo Bob, vuole acquistare, su un sito di e-commerce, il libro "La dieta della Pizza", al prezzo di 15$. Inserisce il libro nel carrello virtuale e completa l'ordine, selezionando "consegna espressa" e come metodo di pagamento "carta di credito".

Questa azione genera automaticamente due obblighi distinti, ma correlati:

- Bob ha l'obbligo di effettuare il pagamento della cifra richiesta con la sua carta di credito entro un giorno dall'acquisto;
- Lisa (colei che vende il prodotto) ha l'obbligo di spedire il libro a partire dal momento in cui Bob effettua il pagamento, fino a 3 giorni dal momento dell'acquisto.

5.2. Inserimento dei dati nell'ontologia

All'avvio, l'applicazione mette a disposizione quattro azioni: inserimento di nuovi dati, effettuare interrogazioni, salvare le modifiche oppure terminare l'esecuzione dell'applicazione. Per prima cosa è necessario inserire i due agenti “Bob” e “Lisa”. Selezionando “*new*” è possibile inserire nuovi dati nell'application ontology (Figura 5.1)

Una volta inserito il nome dell'agente si ritorna al menu principale e si può quindi procedere con nuove operazioni. Selezionando “*save*” è possibile salvare le modifiche nel file e aprirlo con l'editor Protégé. La figura 5.2 riporta gli individui inseriti all’interno dell’ontologia, dove gli individui evidenziati in grassetto sono quelli inseriti con l'applicazione Java, mentre quelli non evidenziati sono ereditati dalle ontologie importate.

```
Welcome!
write "new" to insert new data in the ontology
write "query" to query the ontology
write "exit" to save and quit the program
write "save" to save the changes

->new
What kind of data do you want to insert?
1 - Agent
2 - Event
3 - Deontic Relation
1
Insert new Agent

Insert a name: Bob
<https://dl.dropboxusercontent.com/u/12528226/obligation/AppOntology.owl#agent-1395849797421>
```

Figura 5.1 – Inserimento di Agenti nell’ontologia.

Figura 5.2 – Risultato dell’operazione.

Ora possiamo inserire i due obblighi derivanti dall'acquisto. Dal menu di inserimento selezionare *Deontic Relation.* Viene chiesto di scegliere tra un obbligo o un divieto e, successivamente, vengono richiesti gli agenti che sono coinvolti: è possibile selezionare un agente già esistente nell'ontologia, oppure inserirne uno nuovo (Figura 5.3).

```
Welcome!
write "new" to insert new data in the ontology
write "query" to query the ontology
write "exit" to save and quit the program
write "save" to save the changes

->new
What kind of data do you want to insert?
1 - Agent
2 - Event
3 - Deontic Relation
3
Press 1 to enter a new Obligation or 2 to enter a new Prohibition:
--> 1
Insert new Obligation
Select the Debtor:
Press 1 to select an agent in the ontology or 2 to create a new agent:
--> 1
Press the correct number:
1: https://dl.dropboxusercontent.com/u/12528226/obligation/AppOntology.owl#agent-1395850740525 hasName: Lisa
2: https://dl.dropboxusercontent.com/u/12528226/obligation/AppOntology.owl#agent-1395849797421 hasName: Bob
2
Select the Creditor:
Press 1 to select an agent in the ontology or 2 to create a new agent:
--> 1
Press the correct number:
1: https://dl.dropboxusercontent.com/u/12528226/obligation/AppOntology.owl#agent-1395850740525 hasName: Lisa
2: https://dl.dropboxusercontent.com/u/12528226/obligation/AppOntology.owl#agent-1395849797421 hasName: Bob
1
```

Figura 5.3 – Creazione di una Relazione Deontica: selezione Agenti.

In seguito, viene chiesto di inserire una Descrizione di evento che rappresenta il contenuto dell'obbligo, a scelta tra un pagamento o una consegna. Scegliendo un pagamento, ad esempio, bisogna specificare il tipo tra quelli appartenenti alla gerarchia dei pagamenti, l'ammontare e gli agenti coinvolti (Figura 5.4).

Vengono poi richieste le Descrizioni per la condizione di attivazione e di terminazione. In questi due casi è possibile scegliere tra la descrizione di un'azione (in modo analogo al pagamento appena inserito) oppure un evento temporale, specificando dunque una data e un'ora precise (Figura 5.5).

```
Insert the content of the obligation:
Insert new EventDescription

Select a type:
1 - Payment
2 - Delivery
--> 1
Select the subtype:
p - Payment
c - Cash
b - BankTransfer
a - Allowance
pc - Payment Card
cc - Credit Card
dc - Debit Card
--> pc
Insert the Amount: 15
Select the beneficiary:
Press 1 to select an agent in the ontology or 2 to create a new agent:
--> 1
Press the correct number:
1: https://dl.dropboxusercontent.com/u/12528226/obligation/AppOntology.owl#agent-1395850740525 hasName: Lisa
2: https://dl.dropboxusercontent.com/u/12528226/obligation/AppOntology.owl#agent-1395849797421 hasName: Bob
1
Select the donor:
Press 1 to select an agent in the ontology or 2 to create a new agent:
--> 1
Press the correct number:
1: https://dl.dropboxusercontent.com/u/12528226/obligation/AppOntology.owl#agent-1395850740525 hasName: Lisa
2: https://dl.dropboxusercontent.com/u/12528226/obligation/AppOntology.owl#agent-1395849797421 hasName: Bob
2
```

Figura 5.4 – Creazione di una Relazione Deontica: descrizione dell'Azione.

```
Insert the Activation condition:
Press 1 for a TimeEventDescription or 2 for an ActionDescription:
-->
1
Insert new TimeEventDescription

Insert date and time (aaaa/mm/gg hh:mm:ss) ->2014/03/26 18:00:00
Insert the Termination condition:
Press 1 for a TimeEventDescription or 2 for an ActionDescription:
-->
1
Insert new TimeEventDescription

Insert date and time (aaaa/mm/gg hh:mm:ss) ->2014/03/27 18:00:00
```

Figura 5.5 – Creazione di una Relazione Deontica: condizioni di Attivazione e di Terminazione.

Una volta terminato e salvato, in Protégé si trovano gli individui mostrati nelle figure 5.6 e 5.7. In modo analogo è possibile definire l'obbligo di Lisa di spedire il libro a Bob, dal momento in cui avviene il pagamento fino a 3 giorni dalla data di acquisto.

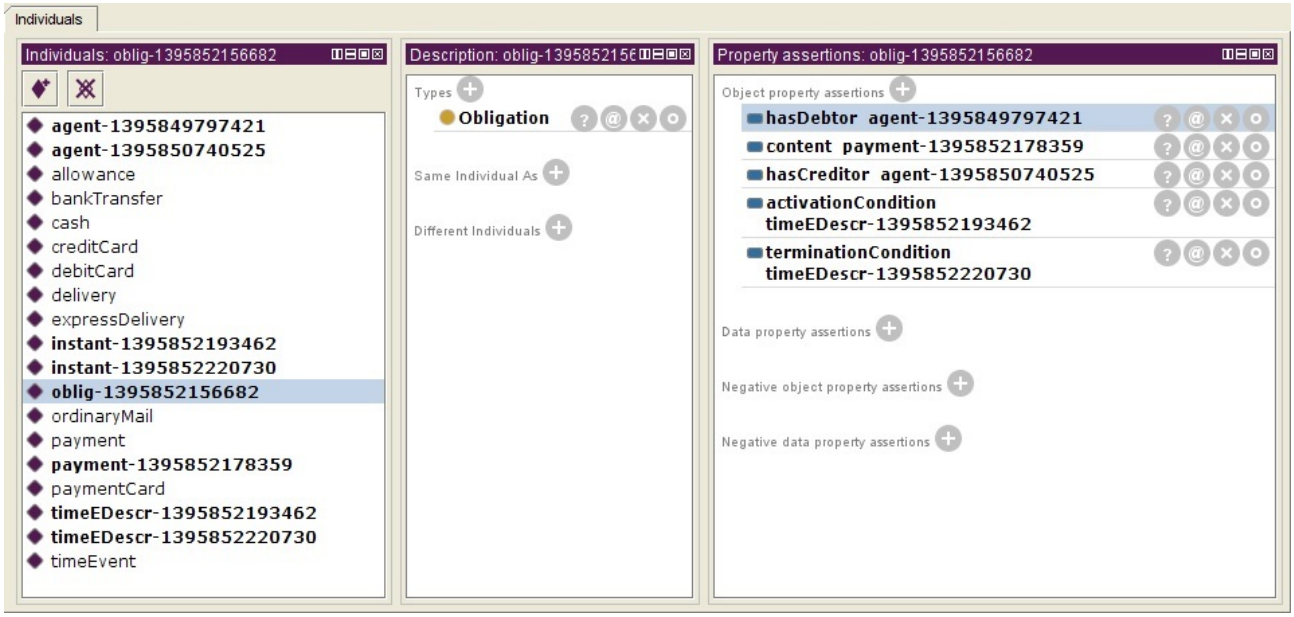

Figura 5.6 – Obligation.

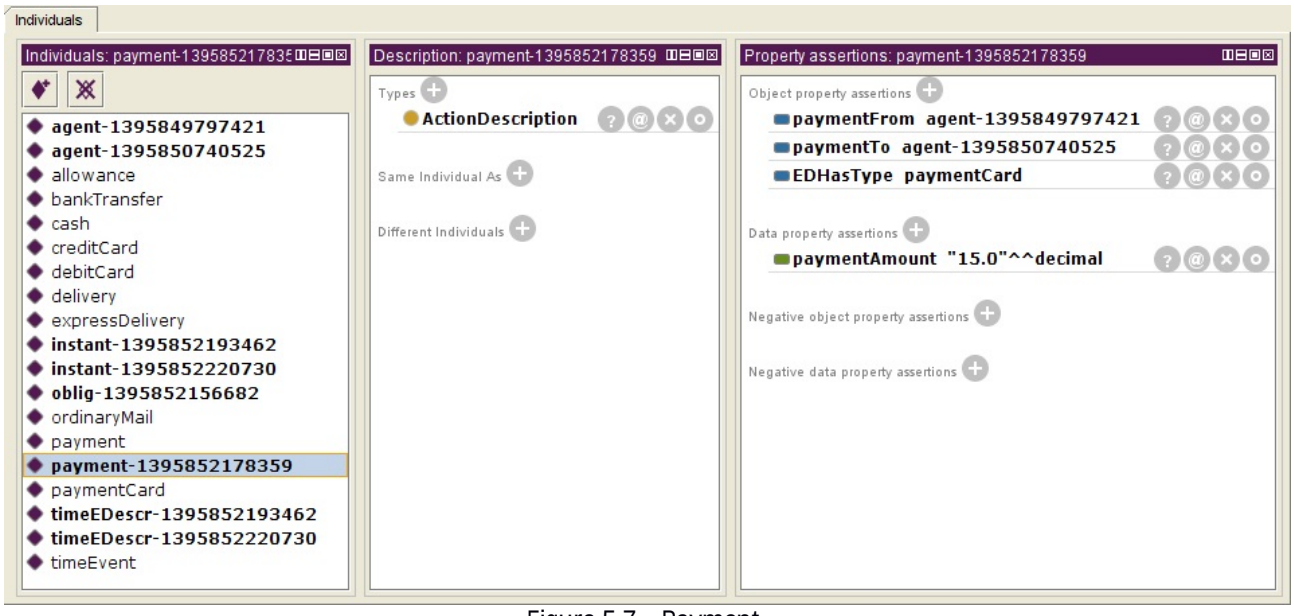

Figura 5.7 – Payment.

Una volta trascorso l'istante indicato dall'evento di attivazione, l'applicazione crea un evento temporale timevent che, a sua volta, crea la parte temporale Activated dell'obbligo (Figura 5.8).

Quando una descrizione di evento temporale si riferisce ad un istante già trascorso l'applicazione crea un evento temporale concreto, che realizza quella descrizione. Questo nuovo evento attiva l'obbligo di Bob: l'applicazione genera una parte temporale di classe Activated, riferita all'individuo che rappresenta l'obbligo e a un altro individuo che rappresenta l'intervallo che inizia nel momento in cui si verifica questo evento.

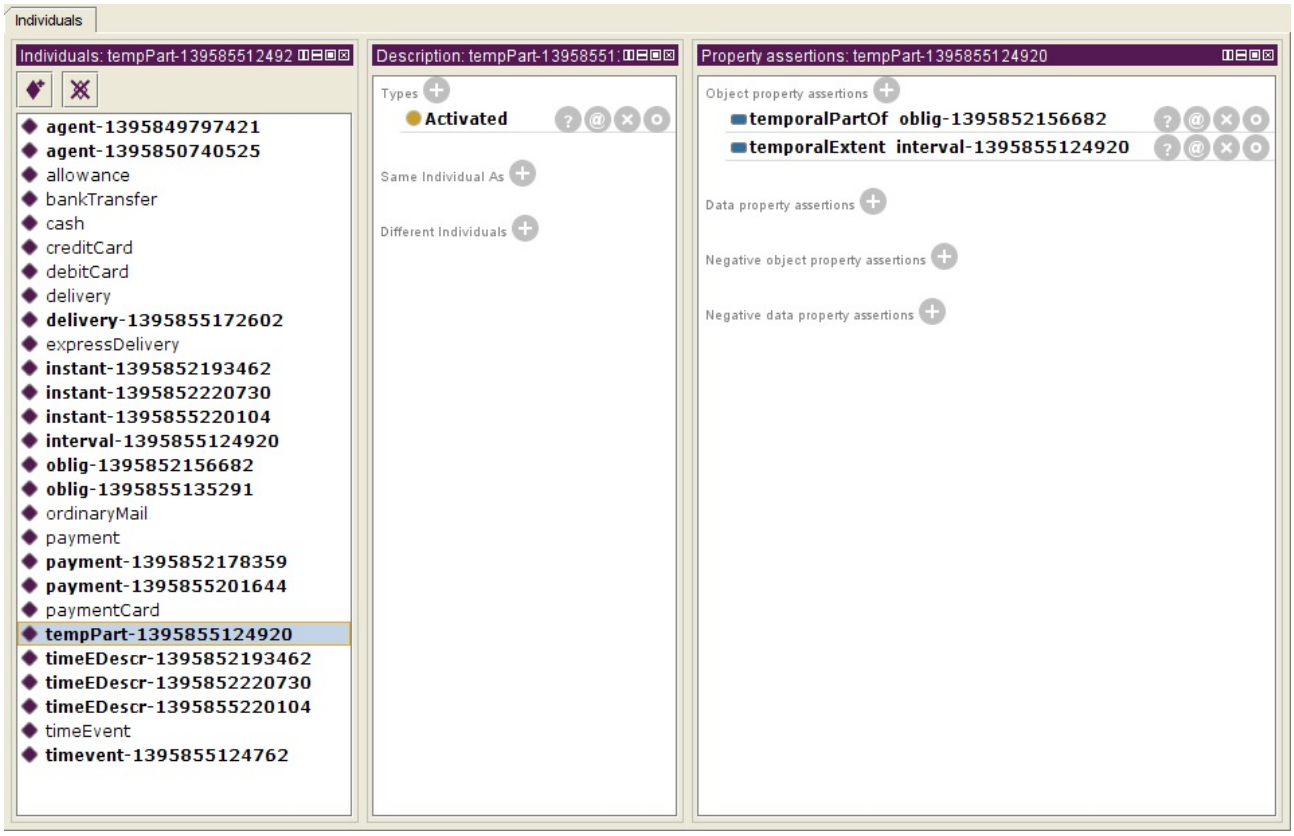

Figura 5.8 – Parte temporale Activated.

Ora che l'obbligo è stato attivato può diventare soddisfatto qualora si verifichi l'azione corrispondente, oppure può essere violato nel caso in cui avvenga l'evento di terminazione. In questo secondo caso, analogamente all'attivazione, viene creato un TimeEvent, che genera la parte temporale Violated dell'obbligo, come mostrato nella figura 5.9.

Vediamo invece cosa accade se si verifica un'azione idonea a soddisfare l'obbligo. Per inserire un evento concreto nell'ontologia selezionare la voce Event dal menu di inserimento. I dati da inserire sono: l'Agent che compie l'azione, una descrizione dell'azione (come per il contenuto dell'obbligo) e la data e l'ora in cui si è verificata l'azione. Questo crea un individuo per la descrizione dell'azione, un individuo che rappresenta l'istante in cui avviene e un individuo di classe Event che indica l'azione concreta.

La descrizione dell'azione appena inserita è sussunta da quella descritta dall'obbligo (ma non viceversa), poiché coinvolge gli stessi agenti, riguarda lo stesso ammontare di denaro, ed è di un sottotipo di quella contenuta nell'obbligo.

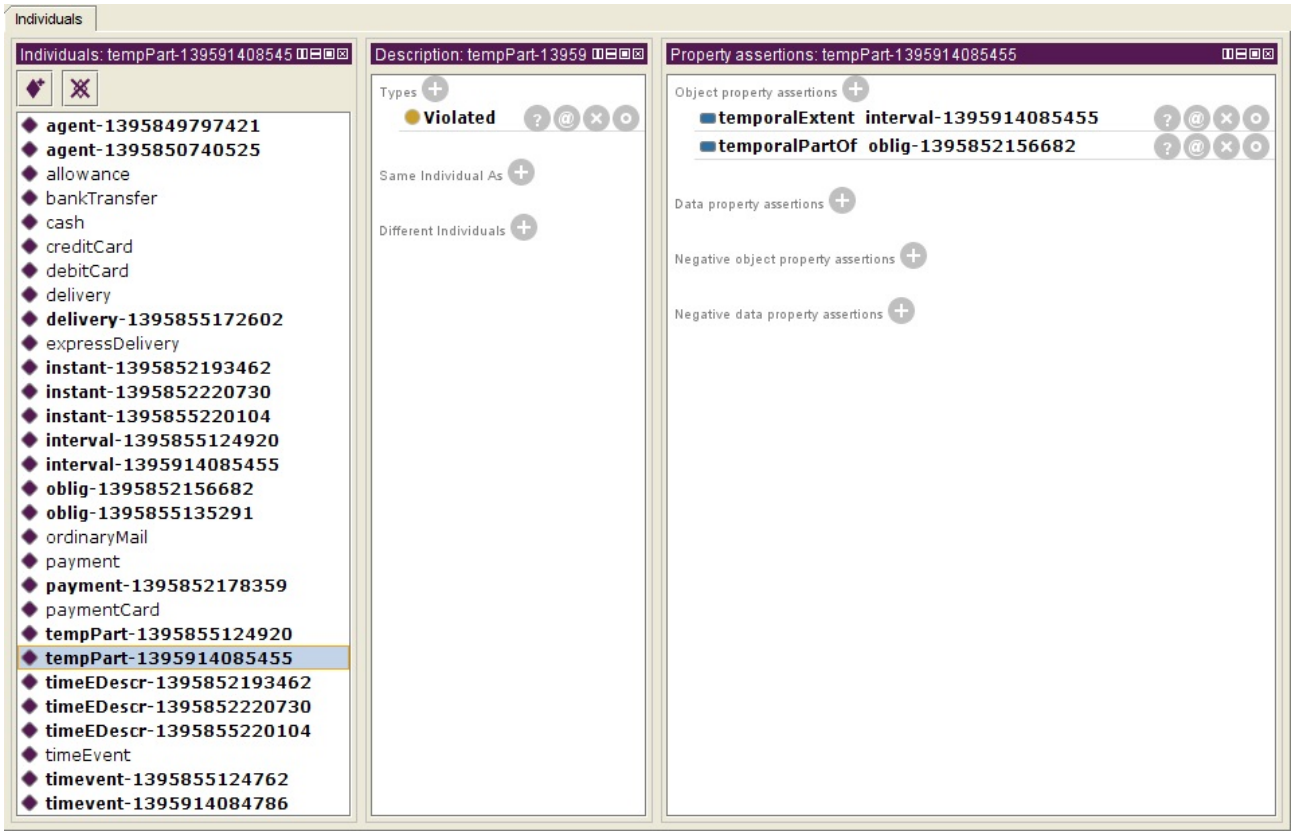

Figura 5.9 – Parte temporale Violated.

Tramite il reasoner, dunque, è possibile dedurre che questa descrizione possiede la proprietà EDSubsumedBy con sé stessa (poiché la proprietà è riflessiva) e con la descrizione più generale. Attivando il reasoner in Protégé, infatti, si vede che i due individui payment sono correlati dalla proprietà EDSubsumedBy (Figura 5.10).

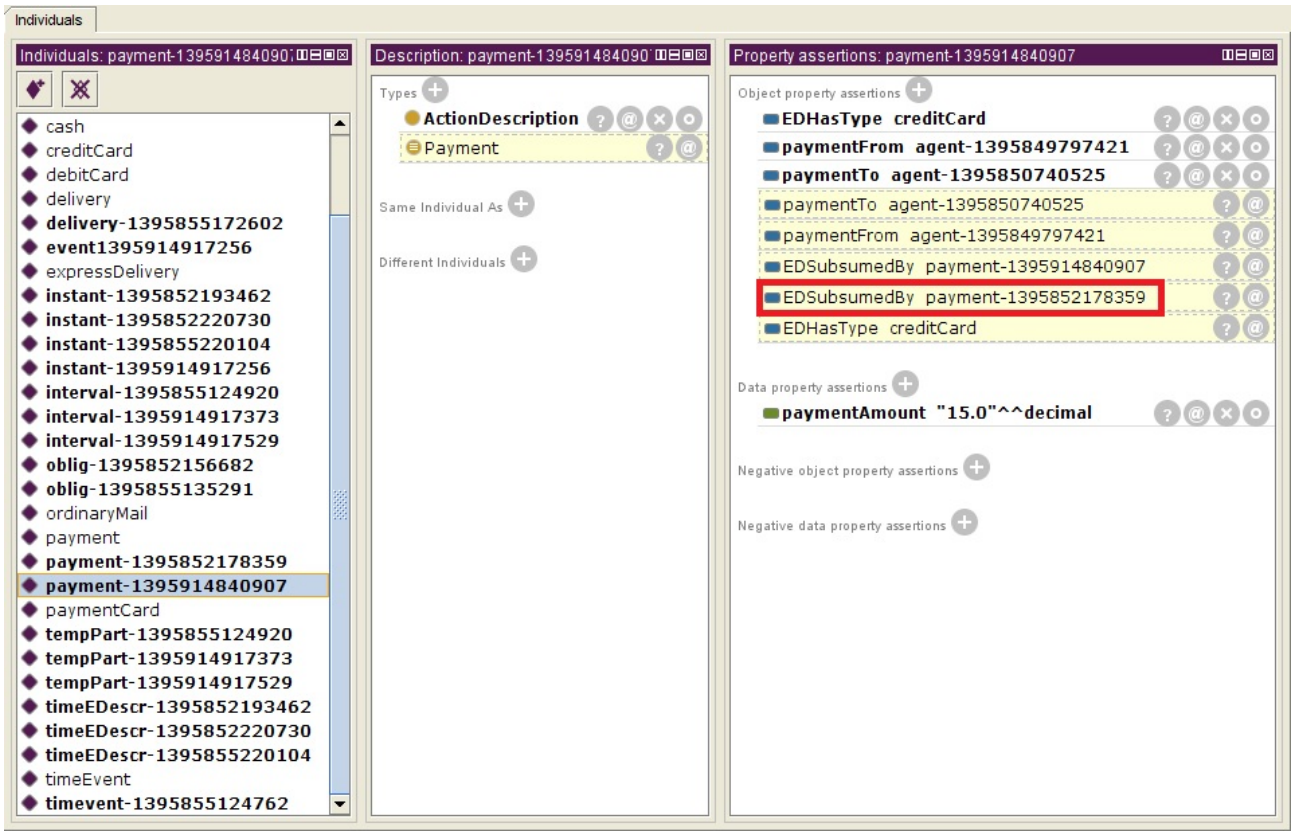

Figura 5.10 – Pagamento sussunto.

L'inserimento dell'azione provoca la creazione di una parte temporale Fulfilled per l'obbligo di Bob, con le relative proprietà, e una parte temporale Activated per l'obbligo di Lisa di effettuare la consegna (Figura 5.11).

Analogamente, l'obbligo di Lisa può essere soddisfatto se essa compie un'azione idonea, altrimenti l'obbligo verrà violato. L'applicazione creerà le parti temporali che rappresentano la corretta evoluzione degli stati delle relazioni deontiche inserite, indicando gli intervalli di tempo in cui questi stati sono validi.

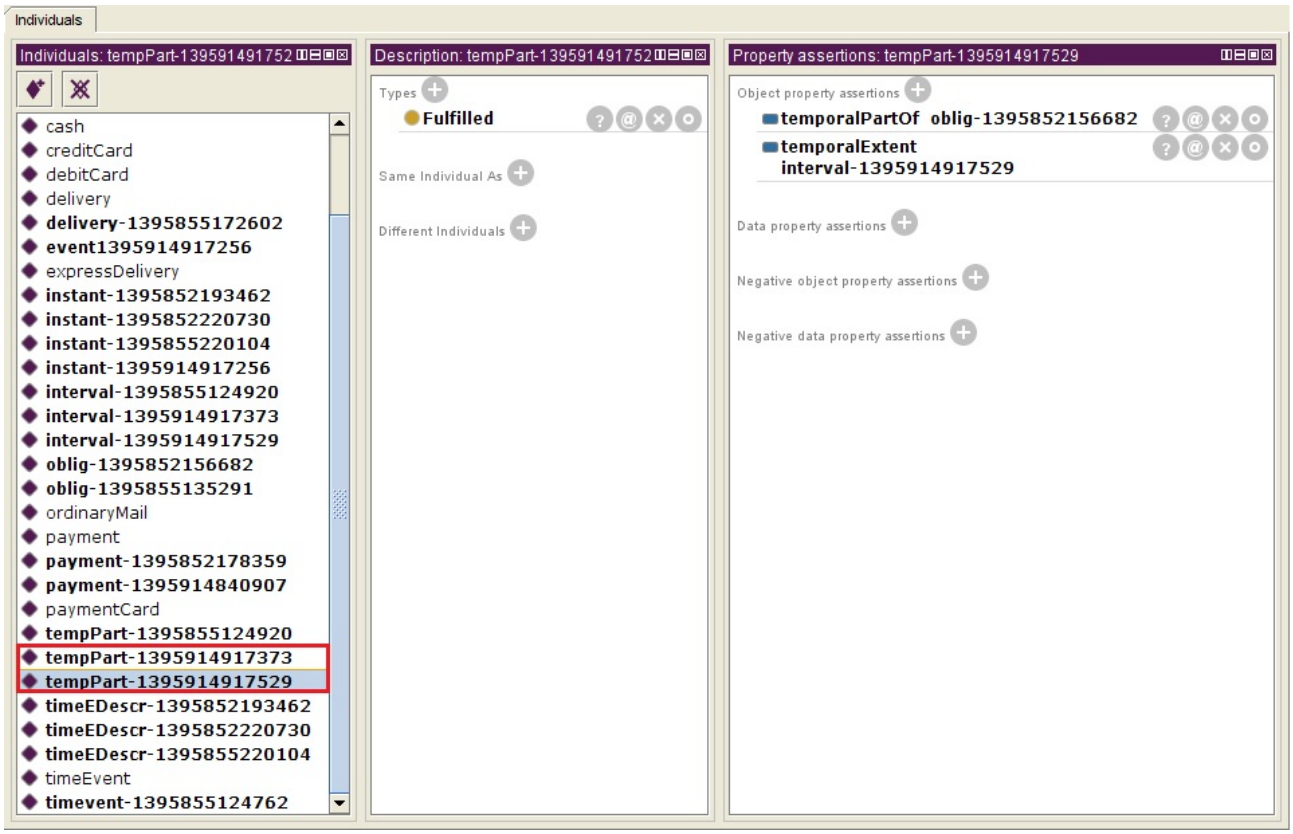

Figura 5.11 – Parte temporale Fulfilled.

CAPITOLO 6

CONCLUSIONI

6.1. Analisi degli obiettivi del progetto

La rappresentazione in ontologie OWL di relazioni deontiche e degli stati che assumono, in seguito al verificarsi di eventi, è un problema ricorrente in situazioni in cui vi sono interazioni tra agenti e in particolare in ambiti commerciali o legali.

Il modello proposto in questo progetto permette di definire obblighi e divieti adattandosi a qualsiasi tipo di evento preso in considerazione grazie alla definizione generica di evento divisa in tre livelli concettuali. Il livello di tipo, il livello di descrizione e il livello di realizzazione. Grazie a questa divisione è semplice definire la relazione di sussunzione tra una descrizione di evento più specifica e una descrizione più generale, permettendo di considerare molteplici tipi di azione come idonee a soddisfare un singolo obbligo. La definizione di tipi di evento utilizzabili in applicazioni reali è dunque semplice attraverso l'estensione del modello proposto e la definizione della regola di sussunzione specifica dell'ambito applicativo.

Una caratteristica importante del modello è la separazione concettuale della semantica dai dati applicativi. Ciò facilita la diffusione del modello e il riutilizzo dello stesso in contesti anche molto diversi tra loro. Rispetto ad altri modelli questo permette una chiara separazione, rappresentata dalla struttura a tempio greco riportata nel Capitolo 2. È semplice dunque realizzare ontologie che contengono solamente i dati dell'applicazione importando i concetti e la semantica associata a questi dati da varie ontologie di dominio e dalla Top Ontology.

6.2. Sviluppi futuri

Nel capitolo 3 abbiamo illustrato vari tipi di obbligo e divieto, trattando però solo il tipo più semplice, gli obblighi singoli: gli altri tipi di obbligo, infatti, possono essere definiti tramite più obblighi singoli oppure, grazie alla dualità, tramite divieti equivalenti.

Un possibile sviluppo della top ontology è quello di definire i concetti di obbligo ripetuto e obbligo complesso attraverso un certo numero di obblighi singoli e specificare le relazioni con questi ultimi: da un obbligo ripetuto, dunque, derivano tanti obblighi singoli con periodi temporali distinti, ma con la stessa descrizione di evento; un obbligo complesso, invece, definisce tanti obblighi singoli con descrizioni diverse, ma con lo stesso periodo di tempo o le stesse condizioni di attivazione e terminazione. In entrambi i casi è necessario definire il modo in cui l'obbligo generale viene soddisfatto oppure violato. Intuitivamente, se tutti gli obblighi singoli sono stati soddisfatti allora l'obbligo ripetuto/complesso è soddisfatto; al contrario, se almeno uno degli obblighi singoli risulta violato allora lo è anche quello generale.

Un caso diverso sono gli obblighi continui: in alcuni casi è possibile trasformarli in divieti, ma non sempre, poiché quando si devono descrivere obblighi su azioni continue la dualità non funziona. Questo introduce il problema già a livello degli eventi perché non è possibile rappresentare eventi continui con il modello attuale. Una possibile soluzione potrebbe essere quella di rappresentare un'azione continua con due eventi che ne rappresentino l'inizio e la fine.

Un obbligo su un'azione continua può dunque essere rappresentato tramite due obblighi semplici e un divieto:

- L'obbligo singolo di iniziare in un determinato istante l'azione o, in alternativa, di averla già iniziata precedentemente;
- L'obbligo singolo di terminarla in un istante successivo al termine dell'obbligo continuo;
- Il divieto di terminarla nell'intervallo tra questi due istanti.

Un possibile sviluppo riguardo la definizione di descrizioni di eventi è quello di rappresentare anche un secondo tipo di evento temporale, ossia eventi di durata. Potrebbe essere utile definire la terminazione di un obbligo dopo un determinato periodo di tempo a partire dall'attivazione dello stesso. Quando la condizione di attivazione dell'obbligo è un evento temporale definito da una data ed un'ora precise, l'evento di terminazione può essere facilmente calcolato ed espresso nello stesso modo, ma se l'evento di attivazione è rappresentato dalla descrizione di un'azione, non è possibile determinare l'istante di terminazione fino a quando l'azione di attivazione non si verifica. Per casi come questo si potrebbe definire una classe DurationEvent, sottoclasse di EventDescription, espressa tramite il riferimento ad un'altra descrizione di evento che funge da istante di partenza e un attributo che rappresenta la durata dell'intervallo.

Altri possibili sviluppi dell'ontologia riguardano la correlazione tra obblighi distinti, rendendo possibile cancellare in maniera automatica un obbligo se un'altra relazione deontica viene violata. Nell'esempio mostrato nel capitolo 5, gli obblighi sono correlati, ma non è possibile descrivere questa correlazione per poter cancellare l'obbligo di Lisa nel caso in cui quello di Bob venga violato. Se l'obbligo di Bob venisse violato, infatti, l'obbligo di Lisa potrebbe non essere mai attivato o, peggio ancora, potrebbe essere attivato erroneamente da un'azione idonea, ma che avviene quando l'obbligo è ormai concluso. Per evitare questo problema è possibile cancellare manualmente l'obbligo di Lisa, evitando così che venga attivato: dato però che spesso la validità di alcuni obblighi o divieti dipende dall'esito di altre relazioni deontiche, sarebbe utile rendere il processo di cancellazione automatico.

Successivamente si potrebbero formalizzare concetti quali contratti e accordi legali che implicano la definizione di molteplici obblighi correlati: si pensi ad esempio a sanzioni in caso di violazione o, viceversa, a premi in caso contrario, e ad eventi che generano obblighi quali promesse o l'atto di siglare un accordo.

APPENDICE A

TOP ONTOLOGY NEL DETTAGLIO: CLASSI, PROPRIETÀ E ATTRIBUTI

In questa prima appendice presenteremo nel dettaglio gli elementi che compongono la struttura dell'ontologia con una breve descrizione di ciascuno di questi e delle relative caratteristiche. Descriveremo dapprima le classi con la relativa gerarchia, successivamente proprietà ed attributi, per finire poi con le regole SWRL che definiscono il cambiamento di stato delle relazioni deontiche e degli eventi associati.

A.1. Classi

Agent	Rappresenta individui che possono essere soggetti ad obblighi o divieti e che sono in grado di compiere azioni.
TemporalEntity	Rappresenta un'informazione temporale; è unione disgiunta di Instant e Interval.
Instant	Individui che rappresentano istanti temporali descritti da un attributo di tipo xsd:DateTime.

Interval	Possiedono due attributi che descrivono gli istanti di inizio e fine dei periodi temporali.
EventType	Individui che reificano i concetti di tipi di evento e che costituiscono una gerarchia attraverso la proprietà di sussunzione.
EventDescription	Definiscono le proprietà di un evento senza però rappresentare la realizzazione dello stesso, escludendo dunque l'istante in cui avviene l'evento e, nel caso di azioni, l'agente che lo compie. Si riferisce ad un EventType con la proprietà hasType.
ActionDescription	Descrizione di azioni.
TimeEventDescription	Descrizione di eventi temporali. Questi individui hanno tipo *timeEvent* (un individuo di classe EventType).
Event	Realizzazione concreta di una descrizione di evento a cui si riferisce tramite la proprietà *realize;* possiede inoltre un riferimento all'istante in cui avviene.
Action	Realizzazione di un'azione: oltre all'istante e alla descrizione comprende anche l'agente che la compie tramite la proprietà hasActor.
TimeEvent	Realizzazione di un TimeEventDescription. Disgiunta dalla classe Action.
DeonticRelation	Rappresenta i vincoli di obbligo e di divieto.
Obligations	Individui che descrivono i vincoli positivi con i relativi agenti coinvolti, le condizioni di attivazione e terminazione e la descrizione dell'evento riferito dall'obbligo.
Prohibitions	Analogamente agli individui di classe Obligations descrive agenti ed eventi che intervengono nel ciclo di vita dei divieti.

DRTemporalParts	Comprende tutti gli stati assunti dalle relazioni deontiche; è unione disgiunta delle classi Activated, Canceled, Fulfilled, Violated.
Activated	Rappresenta, in un determinato intervallo temporale, lo stato attivo di una relazione deontica.
Canceled	Rappresenta una relazione deontica non più valida.
Fulfilled	Rappresenta una relazione deontica soddisfatta.
Violated	Rappresenta una relazione deontica violata.

Nella figura A.1 si può notare la gerarchia delle classi della Top Ontology. La sottoclassi di Thing sono tutte disgiunte tra loro.

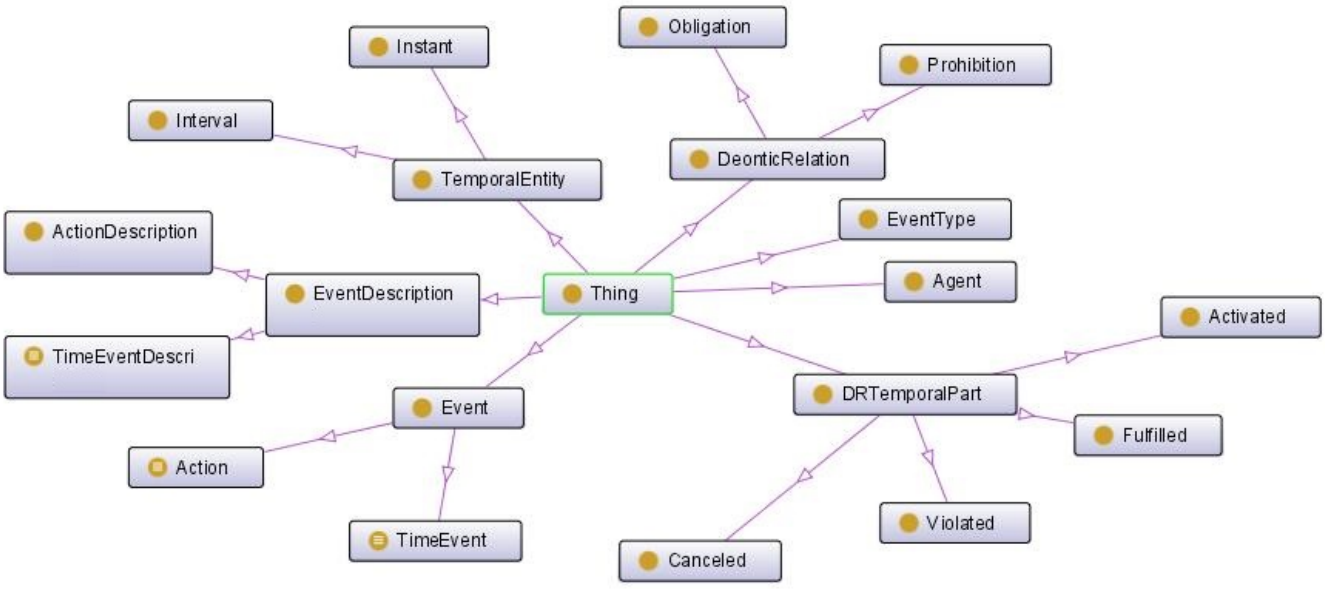

Figura A.1 – Gerarchia delle classi

A.2. Proprietà ed Attributi

activate	Dominio: Event Codominio: DeonticRelation Un evento che soddisfa la condizione di attivazione di una relazione deontica è associato ad essa tramite activate.
activationCondition	Dominio: DeonticRelation Codominio: EventDescription Ogni relazione deontica ha una condizione di attivazione che è rappresentata da una descrizione di evento.
content	Dominio: DeonticRelation Codominio: ActionDescription La descrizione dell'evento che soddisfa un obbligo o divieto è indicata dalla proprietà content.
createTemporalPart	Dominio: Event Codominio: DRTemporalPart Se un evento è responsabile del cambio di stato di una relazione deontica, si dice che crea la relativa temporal part.
EDHasType	Dominio: EventDescription Codominio: EventType Ogni descrizione di evento ha un tipo rappresentato da un individuo di classe EventType.
EDSubsumedBy	(Symmetric) Dominio: EventDescription Codominio: EventDescription Una descrizione di evento è sussunta da un'altra quando è possibile sostituire la prima nel cambio di stato di un obbligo.

fulfill	Dominio: Event Codominio: DeonticRelation Un evento che realizza il contenuto di un obbligo è idoneo a soddisfarlo.
happensAt	(Functional) Dominio: Event Codominio: Instant Ogni evento concreto accade ad un preciso istante temporale.
hasActor	(Functional) Dominio: Action Codominio: Agent Ogni azione si riferisce all'agente che la compie.
hasCreditor	Dominio: DeonticRelation Codominio: Agent Un obbligo può avere un agente verso cui l'azione deve essere compiuta e viceversa per i divieti.
hasDebtor	Dominio: DeonticRelation Codominio: Agent L'agente responsabile della relazione deontica è detto *debitore.*
hasEnd	Dominio: Interval Codominio: xsd:dateTime Istante finale di un intervallo.
hasName	Dominio: Agent Codominio: xsd:string Nome di un agente.

hasStart	Dominio: Interval Codominio: xsd:dateTime Istante iniziale di un intervallo.
hasTime	Dominio: Instant Codominio: xsd:dateTime Data e ora a cui si riferisce un istante.
realize	Dominio: Event Codominio: EventDescription Un evento concreto si riferisce ad una descrizione di evento.
subtypeOfEvent	(Transitive) Dominio: EventType Codominio: EventType Grazie a questa proprietà è possibile definire la gerarchia di tipi.
temporalExtent	(Functional) Dominio: DRTemporalPart Codominio: Interval Ogni parte temporale ha un intervallo di tempo di validità.
temporalPartOf	(Functional) Dominio: DRTemporalPart Codominio: DeonticRelation Ogni parte temporale si riferisce ad un'unica relazione deontica di cui rappresenta lo stato.
terminate	Dominio: Event Codominio: DeonticRelation Un evento che soddisfa la condizione di terminazione di un obbligo è idoneo a terminarlo e, a seconda che si tratti di un obbligo oppure di un divieto, lo violerà oppure soddisferà rispettivamente.

terminationCondition	Dominio: DeonticRelation Codominio: EventDescription Ogni relazione deontica è terminata da un evento descritto da un individuo di classe EventDescription.
timeDescription	(Functional) Dominio: TimeEventDescription Codominio: Instant Una descrizione di evento temporale indica l'istante in cui si verifica tale evento.
violate	Dominio: Event Codominio: DeonticRelation Un evento che realizza il contenuto di un divieto è idoneo a violarlo.

A.3. Regole SWRL

Regola di Attivazione

DeonticRelation(?dr), Event(?e), EDSubsumedBy(?evD, ?actD), activationCondition(?dr, ?actD), realize(?e, ?evD)
-> activate(?e, ?dr)

Regola di Terminazione

Activated(?tp), DeonticRelation(?dr), Event(?e), EDSubsumedBy(?evDescr, ?endDescr), createTemporalPart(?eg, ?tp), happensAt(?e, ?if), happensAt(?eg, ?ia), realize(?e, ?evDescr), temporalPartOf(?tp, ?dr), terminationCondition(?dr, ?endDescr), hasTime(?ia, ?ta), hasTime(?if, ?tf), lessThan(?ta, ?tf)
-> terminate(?e, ?dr)

Regola di Violazione di Divieti

Activated(?tp), Agent(?a), Event(?e), Prohibition(?p), EDSubsumedBy(?evDescr, ?pDescr), content(?p, ?pDescr), createTemporalPart(?eg, ?tp), happensAt(?e, ?if), happensAt(?eg, ?ia), hasActor(?e, ?a), hasDebtor(?p, ?a), realize(?e, ?evDescr), temporalPartOf(?tp, ?p), hasTime(?ia, ?ta), hasTime(?if, ?tf), lessThan(?ta, ?tf)
-> violate(?e, ?p)

Regola di Soddisfacimento di Obblighi

Activated(?tp), Agent(?a), Event(?e), Obligation(?o), EDSubsumedBy(?evDescr, ?oblDescr), content(?o, ?oblDescr), createTemporalPart(?eg, ?tp), happensAt(?e, ?if), happensAt(?eg, ?ia), hasActor(?e, ?a), hasDebtor(?o, ?a), realize(?e, ?evDescr), temporalPartOf(?tp, ?o), hasTime(?ia, ?ta), hasTime(?if, ?tf), lessThan(?ta, ?tf)
-> fulfill(?e, ?o)

Regola di Coerenza di TimeEvent e relativa TimeEventDescription

TimeEvent(?te), TimeEventDescription(?td), realize(?te, ?td), timeDescription(?td, ?i) -> happensAt(?te, ?i)

Riflessività locale della proprietà *subtypeOfEvent*

EventType(?t) -> subtypeOfEvent(?t, ?t)

Riflessività locale della proprietà *EDSubsumedBy*

EventDescription(?e) -> EDSubsumedBy(?e, ?e)

APPENDICE B

DEFINIZIONE DI ONTOLOGIE DI DOMINIO: PAYMENTS E DELIVERIES

In quest'appendice presenteremo il processo logico di definizione di un'ontologia di dominio che importa e sfrutta i concetti di relazioni deontiche. Lo sviluppo di ontologie di dominio segue due punti fondamentali: la creazione di una gerarchia dei tipi e la determinazione di una regola di sussunzione dei tipi.
Il primo passo della descrizione di eventi è proprio la definizione del tipo di evento. È importante dunque avere a disposizione individui che li rappresentano e come sono correlati tra loro questi individui.

La proprietà
subtypeOfEvent: EventType → EventType
permette di comporre una struttura gerarchica di tipi grazie alla transitività, perciò è semplice verificare che un'azione abbia un tipo più specifico di un'altra.

Le descrizioni di eventi, oltre a riferirsi ad un tipo all'interno della gerarchia, sono spesso identificate da alcuni "parametri" (attributi e/o proprietà) e, proprio in base a questi, si definisce la relazione di sussunzione tra due eventi. Non è sufficiente che due eventi abbiano un tipo "confrontabile", ma è necessario anche che i parametri degli eventi siano equivalenti e ciò viene controllato da una regola di sussunzione.

La struttura di una regola di sussunzione è analoga alla seguente:

MyEventDescription(?e1), MyEventDescription(?e2),
EDHasType(?e1,?t1),EDHasType(?e2,?t2),subTypeOfEvent(?t1,?t2),
MyAttribute(?e1,?a1),MyAttribute(?e2,?a1),
... (other attributes)
MyProperty(?e1,?p1),MyProperty(?e2,?p1),
... (other properties)
➔ EDSubsumedBy(?e1,?e2)

Vediamo ora alcuni esempi di ontologie di dominio.

B.1. Payments

L'ontologia che rappresenta i pagamenti contiene una classe Payment che racchiude tutte le descrizioni di questo tipo. Payment è una sottoclasse di EventDescription e da questa deriva le proprietà EDHasType ed EDSubsumedBy. Oltre a queste è necessario definire proprietà ed attributi per rappresentare i "parametri" di un generico pagamento. Perciò si aggiungono le proprietà:

paymentFrom: Payment -> Agent
paymentTo: Payment -> Agent

e l'attributo:

paymentAmount: Payment -> xsd:Decimal

Successivamente si definiscono alcuni EventType che rappresentano i vari tipi di pagamento, e come sono correlati attraverso la proprietà subTypeOfEvent (Figura B.1)

Infine si definisce la regola di sussunzione:

```
Payment(?p1), Payment(?p2),
EDHasType(?p1, ?t1), EDHasType(?p2, ?t2), subtypeOfEvent(?t1, ?t2),
paymentFrom(?p1, ?from), paymentFrom(?p2, ?from),
paymentTo(?p1, ?to), paymentTo(?p2, ?to),
paymentAmount(?p1, ?a), paymentAmount(?p2, ?a)
 -> EDSubsumedBy(?p1, ?p2)
```

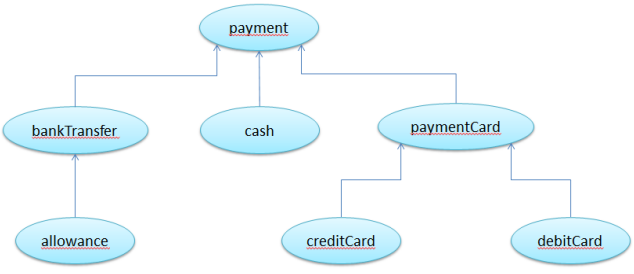

Figura B.1 – Gerarchia dei pagamenti

B.2. Deliveries

L'ontologia che rappresenta le consegne contiene una classe Delivery che racchiude tutte le descrizioni di questo tipo. Delivery è una sottoclasse di EventDescription e da questa deriva le proprietà EDHasType ed EDSubsumedBy. Oltre a queste è necessario definire proprietà ed attributi per rappresentare i "parametri" di una generica consegna. Perciò si aggiungono le proprietà:

deliverySender: Delivery -> Agent
deliveryRecipient: Delivery -> Agent

e l'attributo:

deliveryContent: Delivery -> xsd:String

Successivamente si definiscono alcuni EventType che rappresentano i vari tipi di consegna e come sono correlati attraverso la proprietà subTypeOfEvent (Figura B.2)

Infine si definisce la regola di sussunzione:

```
Delivery(?d1), Delivery(?d2) ,
EDHasType(?d1, ?t1), EDHasType(?d2, ?t2), subtypeOfEvent(?t1, ?t2),
deliveryRecipient(?d1, ?r), deliveryRecipient(?d2, ?r),
deliverySender(?d1, ?s), deliverySender(?d2, ?s),
deliveryContent(?d1, ?c), deliveryContent(?d2, ?c)
 -> EDSubsumedBy(?d1, ?d2)
```

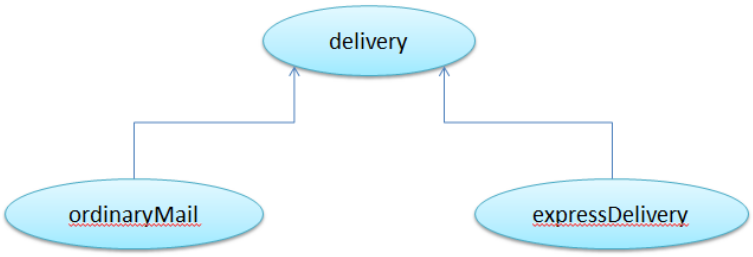

Figura B.2 – Gerarchia delle consegne

BIBLIOGRAFIA

[1] P. Hitzler, M. Krötzsch and S. Rudolph, Foundations of Semantic Web Technologies, 2009, Chapman & Hall/CRC.

[2] World Wide Web Consortium (W3C). Semantic Web [Online]. Available:http://www.w3.org/standards/semanticweb/

[3] Hendler, J., "Web 3.0 Emerging," Computer , vol.42, no.1, pp.111,113, Jan. 2009

[4] N. Fornara and M. Colombetti, "Ontology and Time Evolution of Obligations and Prohibitions using Semantic Web Technology"

[5] World Wide Web Consortium (W3C). OWL 2 Web Ontology Language [Online]. Available:http://www.w3.org/TR/owl2-overview/

[6] World Wide Web Consortium (W3C). SWRL: A Semantic Web Rule Language Combining OWL and RuleML [Online]. Available:http://www.w3.org/Submission/SWRL/

[7] Ceri, S.; Gottlob, G.; Tanca, L., "What you always wanted to know about Datalog (and never dared to ask)," Knowledge and Data Engineering, IEEE Transactions on , vol.1, no.1, pp.146,166, Mar 1989

[8] World Wide Web Consortium (W3C). SPARQL 1.1 Overview[Online]. Available:http://www.w3.org/TR/sparql11-overview/

[9] World Wide Web Consortium (W3C). RDF CURRENT STATUS[Online]. Available:http://www.w3.org/standards/techs/rdf#w3c_all

[10] Sotiris Batsakis, Euripides G. M. Petrakis, "Representing Temporal Knowledge in the Semantic Web: The Extended 4D Fluents Approach", Combinations of Intelligent Methods and Applications, Springer Berlin Heidelberg

[11] M. Horridge and S. Bechhofer, "The OWL API: A Java API for OWL ontologies", Semantic Web, Vol. 2, no. 1, pp. 11-21, Jen. 2011