RADAR Project

Documento di specifica

September 2, 2019

1 Visione generale

I mercati finanziari sono uno dei principali contesti applicativi in ambito Information Technology (IT), caratterizzati da volumi elevati di dati generati ogni giorno. Le banche e le istituzioni finanziarie si scambiano continuamente flussi di dati che descrivono le operazioni finanziarie. Tali flussi di dati, provenienti da diverse fonti, alimentano costantemente i sistemi informativi e le loro banche dati.

L'attuale tecnologia dei sistemi informativi è in grado di memorizzare ed elaborare una elevata quantità di dati. Gli amministratori di database e gli esperti di integrazione di sistemi hanno sviluppato soluzioni per ricevere e gestire in modo efficiente questi flussi di dati, ottimizzando il più possibile il processo di caricamento degli stessi.

A differenza delle attività tradizionali, tali dati sono essenziali per le attività di reportistica, in quanto necessarie per uso interno, per la revisione contabile e per i processi decisionali. Di fatto, le autorità istituzionali e di regolamentazione, come le banche centrali e le agenzie di rating, impongono agli istituti finanziari di fornire periodicamente report che illustrino l'attuale situazione finanziaria della loro attività, in modo da verificarne la conformità rispetto alla normativa vigente. I collaboratori coinvolti in queste attività devono quindi interpretare sistematicamente i requisiti interni ed esterni sulla base dei dati disponibili.

Uno dei principali punti critici delle organizzazioni riguarda la mancanza di dati integrati e la mancanza di un dizionario di dati standard. E' necessario quindi un significativo intervento manuale per definire le interrogazioni necessarie per estrarre i dati (query) e creare i report. Allo stesso tempo però i dipendenti responsabili della produzione di report in generale non sono a conoscenza degli aspetti tecnici relativi all'archiviazione dei dati stessi e alla loro interrogazione; di conseguenza, è necessaria una continua interazione con tecnici esperti.

Chiaramente, esiste un divario tra utenti e sistemi, che può essere riassunto dalle seguenti domande: "Qual è il significato dei dati? Come potrebbero essere interpretati i dati per soddisfare i requisiti di questi organismi di regolamentazione? Come è possibile definire i rapporti fra esperti di dominio e tecnici? Di fatto, termini e concetti sono spesso definiti in modo informale dalle organizzazioni finanziarie e il loro significato può variare a seconda delle comunità specifiche di tali istituzioni.

NICOLA: FRASE NON CHIARISSIMA

Un ulteriore aspetto critico riguarda il fatto che gli esperti di dominio si basano su una terminologia formata da concetti che possono apparire complessi oltre che interconnessi, oltre a quelli definiti da standard internazionali e sistemi sviluppati per fornire servizi finanziari globali (si veda per esempio la letteratura riferita a sistemi come FIBO e Schema.org [14]).

Una definizione comune dei termini e della loro semantica è quindi essenziale per la generazione di una corrispondente reportistica.

L'attività oggetto del presente progetto considera ed affronta il problema definendo un "ponte" tra le attività da specificare per la generazione di documenti di reportistica, sulla base di concetti e termini tipici di un determinato contesto applicativo (come il mercato finanziario), e il sistema informativo che memorizza i dati da interrogare per produrre i report stessi.

Il framework implementato, RADAR (acronimo di Rich Advanced Design Approach for Reporting) parte dalla nozione di Operational Data Store (ODS) [7, 15], cioè un database che fornisce una visione unificata dei dati, e si pone come parte integrante fra un'ontologia di termini e concetti e l'effettivo schema operativo (e relazionale) dei dati di origine. Una volta definito, lo schema relazionale "RADAR Schema" (così chiamato perché basato sul modello dei dati "RADAR Data Model") fornisce una visione di alto livello dei dati di origine sulla base dei concetti descritti nell'ontologia per lo specifico contesto applicativo. In altre parole, esso fornisce una visione concreta dei concetti sulla base dei dati effettivi memorizzati all'interno dell'ODS su cui lavora il framework per i dipendenti. Ciò permette loro di avere una visione dei dati semanticamente migliore: essi infatti possono utilizzare questo schema per specificare le misure aggregate da inserire nei report, sfruttando i termini provenienti dall'ontologia, e definire in tal modo al meglio i dati da utilizzare per la generazione della documentazione finale (report).

Si noti che, anche se l'approccio proposto è stato applicato al contesto finanziaro, esso è stato definito come un framework generale e, di conseguenza, può essere considerato ed applicato anche in contesti differenti da quello descritto nel presente progetto.

2 Concetti di Base e Cenni Letterari

Nel mercato finanziario, il problema di produrre reportistica per le istituzioni di controllo, come le banche centrali, non è nuovo; tuttavia, il continuo e crescente sviluppo dei Sistemi Informativi ha cambiato lo scenario applicativo. In passato, le istituzioni finanziarie hanno mantenuto, ove possibile, gli stili di formattazione della documentazione da produrre, riducendo al minimo il numero di modifiche da apportare per produrre i report richiesti.

Ciò ha portato ad uno scenario di dati con vari sistemi di reportistica fra loro differenti, senza definire criteri standard di formattazione dei dati, e riducendo, di conseguenza, la qualità complessiva fornita dal servizio finanziario. Come riportato dalla letteratura [4], la Banca per gli Accordi Internazionali (Bank of International Settlements, BIS) ha emanato la direttiva del Comitato di Basilea sulla vigilanza bancaria (BCBS), che mira a regolamentare le organizzazioni nel processo di automazione dei processi di reportistica, al fine di ridurre la dipendenza fra i dati e i report prodotti.

E' necessario tuttavia considerare il fatto che i costi per l'introduzione di automatismi nei processi di reportistica hanno causato un aumento significativo delle difficoltà da parte degli

istituti di credito a conformarsi alla direttiva sopra indicata. Inoltre, la corretta definizione di un modello di reportistica è fortemente correlata alla corretta interpretazione della diversa terminologia utilizzata da tali organizzazioni.

In tale scenario, Petrova e colleghi hanno presentato in [14] un appproccio in cui il Semantic Web possa essere visto come un modo efficace per presentare i dati, considerandoli in un database.. Il Semantic Web è stato formalizzato dagli autori come una descrizione dei concetti, e dei termini corrispondenti, per descrivere i dati e le informazioni memorizzate all'interno di un database.

Le ontologie, il loro linguaggio descrittivo e gli strumenti di gestione, sono tra gli elementi centrali del Semantic Web [4]. La *Financial Industry Business Ontology* (FIBO) [14] specifica il collegamento semantico tra i concetti finanziari, così come le loro descrizioni.

L'approccio presentato in [5] evidenzia come l'industria finanziaria sia in grado di gestire dati e vocabolario associato utilizzando tecnologie semantiche. In particolare, gli autori evidenziano come il lavoro esamini l'approccio seguito da esperti, dipendenti e analisti coinvolti nel reporting finanziario, al fine di identificare i meccanismi sociali e istituzionali applicati per costruire un vocabolario comune standard, utilizzando modelli basati sull'ontologia. Un tale vocabolario comune, standardizzato basato sull'ontologia, dovrà infatti essere alla base della progettazione di Sistemi Informativi (SI) di nuova generazione semanticamente abilitati per l'industria finanziaria.

Un altro aspetto si basa sul fatto che le ontologie sono sviluppate da esperti di dominio al fine di acquisire la conoscenza specifica del dominio considerato [13].

Oltre a FIBO, anche un'altra ontologia, chiamata Schema.org, è diventata recentemente molto popolare. Come riportato in letteratura ([17], [10]), Schema.org è un vocabolario ben definito ed eterogeneo che copre molti domini. Anche se l'adozione di Schema.org è in aumento, la completezza delle annotazioni dei dati collegati è ancora una questione critica, principalmente a causa delle dimensioni del vocabolario, dell'incoerenza semantica che sussiste per alcune annotazioni e della mancanza di indicazioni agli utenti per le classi e le proprietà da considerare. Tale aspetto critico è stato trattato in letteratura da alcuni approcci sviluppati al fine di analizzare il vocabolario nelle applicazioni del mondo reale [11], così come l'estensione al fine di definire annotazioni maggiormente espressive.

2.1 Ontologie e Linked Data

Un'ontologia denota il tentativo di formulare uno schema concettuale esaustivo, rigoroso e gerarchicamente strutturato all'interno di un dato dominio, che può essere utilizzato come fondamento di una base di conoscenza[12, 8, 19]. La concettualizzazione del dominio corrisponde alla modellazione di un fenomeno astratto attraverso l'identificazione di concetti rilevanti per la caratterizzazione del dominio stesso. Ciò avviene attraverso l'esplicita definizione di concetti, proprietà e vincoli che definiscono tale fenomeno come il dominio [21]. In ambito finanziario, ad esempio, il concetto di *finanziamento* e di *piano di ammortamento* sono correlati da precise relazioni. In particolare, un'ontologia è descritta dai seguenti componenti principali:

- *Concetto/Classe* Rappresenta qualsiasi oggetto del mondo reale (fisico o virtuale) che può essere descritto. Può infatti corrispondere ad un compito, una funzione, un'azione,

una strategia di ragionamento o un processo di ragionamento, un oggetto fisico, una nozione o un'idea. I concetti si dividono in varie tipologie. Essi possono essere infatti astratti o concreti, elementari, o derivati dall'aggregazione di altri concetti elementari.

- *Relazione* Rappresenta una correlazione fra i concetti e le classi.
- *Assiomi* Sono affermazioni vere sul dominio descritto dall'ontologia. Sono usate per specificare la semantica dei concetti. Generalmente specificano come possa essere utilizzato il vocabolario dei concetti.

Ogni ontologia può inoltre essere classificata secondo le seguenti categorie principali:

- *Ontologie di "Top-level" (fondamentali, basilari)*: corrispondono ad ontologie interdisciplinari che costituiscono gli elementi basilari del web semantico.
- *Ontologie di dominio*: modellano porzioni specifiche di conoscenza, identificando le entità di interesse e le loro relazioni. In linea di principio, le ontologie specifiche di un dominio si basano su ontologie di alto livello.
- *Ontologie applicative*: definiscono un'estensione di un'ontologia di dominio con scopi molto specifici.

L'approccio inizialmente presentato da Berners Lee in [3] si basa sulla definizione di una base di conoscenza definita solo considerando i concetti, in quanto non vengono inizialmente considerati i dati e i conseguenti aspetti critici legati alla loro gestione. Il principale problema derivante da un tale approccio infatti è legato al fatto che i concetti così definiti non trovano corrispondenza in uno scenario reale. Per questo motivo gli autori propongono un ulteriore approccio in cui vengono considerati dati reali e le loro relazioni, successivamente impiegati per modellare i concetti che definiscono la base di conoscenza.

Attraverso i dati collegati è possibile pubblicare dati sul web in modo leggibile ed interpretabile da una macchina, il cui significato è definito da una stringa di parole e marcatori (triple) per costruire una rete dati appartenente ad un dominio, in grado di essere collegabile ad altri insiemi di dati oltre che ad altri domini sul web. Ciò permette di definire quindi una rete dati globale, i cui contenuti possono essere scambiati ed interpretati dalle macchine (concetti base per il Web semantico [3, 9]). Di conseguenza, i formati per rappresentare le ontologie devono essere considerati in fase di definizione di un'ontologia.

In un tale scenario è possibile affermare che il framework *RADAR* implementato in questo lavoro, a partire dalla nozione di *Operational Data Store* (ODS), si colloca a metà strada tra un'ontologia di termini e concetti e l'effettivo schema operativo (e relazionale) dei dati di partenza. Inoltre, lo schema corrispondente e implementato permette una visione di alto livello dei dati sorgente sulla base dei concetti descritti nell'ontologia per lo specifico contesto applicativo.

Per quanto riguarda i formati di rappresentazione delle ontologie, si ricorda come il W3C specifica che il formato *JSON-LD* [18] è un formato standard per la codifica dei dati collegati usando JSON come framework sintattico di base [6]. In particolare, JSON-LD è progettato intorno al concetto di "contesto": collega le proprietà dell'oggetto in un documento JSON a concetti in un'ontologia. Per mappare la sintassi di JSON-LD in RDF, JSON-LD permette

ai valori di essere costretti a un tipo specifico o di essere taggati con una lingua. Un contesto può essere incorporato direttamente in un documento JSON-LD o inserito in un documento separato e referenziato da documenti diversi.

3 L'Approccio

3.1 Descrizione del Problema

La soluzione del progetto nasce dalla necessitá di generare in maniera semi-automatica i report di sintesi da inviare alle BCE, Banca d'Italia, etc. ottenuti analizzando i dati finanziari posseduti dalle varie banche. Al momento, la generazione del report avviene manualmente in tutte le fasi del processo, dalla definizione delle aggregazioni da calcolare, fino alla generazione del report Excel. Ciò che si vuole realizzare è uno strumento che consenta di gestire il processo di automatizzazione della generazione del report.

Lo strumento deve inoltre consentire la definizione e la gestione di una "base di conoscenza" relativa al contesto partendo dall'ontologia di Schema.org. Tale ontologia è definita da concetti presenti o introdotti ex novo nella fase inziale del framework implementato.

A questo si aggiunge l'aspetto critico che le informazioni di dominio, contestualizzate in uno scenario come ad esempio quello finanziario, non sono attualmente memorizzate su alcun supporto informatizzato, ma concorrono unicamente a formare il know-how dell'esperto di dominio.

Da tutto ciò deriva la necessità di definire un sistema, non solo in grado di automatizzare processi al momento imputati ad esperti umani, ma anche di gestire in maniera ottimizzata le informazioni prodotte, al variare del report richiesto. L'idea di affrontare il problema della gestione dei dati integrati secondo un dizionario standard, rispondendo ai requisiti degli organismi di regolamentazione, parte da una serie di considerazioni preliminari.

Considerando inoltre una visione ad alto livello dei dati, si nota come l'effettiva struttura dei dati sorgente, provenienti dal flusso di alimentazione (e conservati all'interno dei database), spesso presenta nomi di attributi che non indicano chiaramente la loro semantica. Questo è il caso dei nomi non descrittivi: il loro significato è noto infatti agli amministratori dei database, ma non è comprensibile per i dipendenti/analisti. Al contrario, essi necessitano di una visione di alto livello dei dati, facilmente comprensibile e più adatta all'analisi. La semantica dei dati potrebbe essere concettualmente caratterizzata da un'ontologia, permettendo a quest'ultima di fornire un quadro concettuale più chiaro al fine di meglio comprendere i dati riportati.

3.2 Metodologia Sviluppata

La visione generale del sistema prevede due attori principali:

- Esperto di dominio: personale della banca e profondo conoscitore del contesto bancario, delle normative e di come si calcolano i vari campi di un report.
- Tecnico informatico: Conoscitore della struttura interna dell'applicativo, e colui che opera le modifica ad esso.

L'esperto del dominio esplicita al tecnico informatico quali sono i concetti che devono essere implementati e le regole di calcolo. Sarà successivamente compito del tecnico informatico implementarle.

Al fine di definire gli elementi costitutivi del *RADAR Data Model*, viene considerato un esempio, che considera, in particolare, un tipico flusso di dati fra banche. Si suppone di essere una banca b_1.

Una *Cessione* è un contratto firmato dalla banca b_1 con un'altra banca b_2, per mezzo del quale b_1 ha acquistato un *prestito* da b_2; la definizione di una nuova cessione implica che nuovi prestiti siano stati acquistati da b_1.

Ogni mese, i flussi di dati forniscono nuovi *Stati finanziatori* per ogni cessione: ad esempio, il debito residuo, il numero di rate pagate e il numero di rate non pagate. Tutte queste informazioni sono segnalate dallo stato finanziario mensile. Inoltre, ogni mese viene ricevuto il *piano di ammortamento* per il flusso di dati, in quanto varia secondo le operazioni eseguite in merito al finanziamento (ad esempio, se il cliente ha pagato una rata, la voce corrispondente del piano di ammortamento viene rimossa).

La soluzione adottata nel presente lavoro premette di avere una visione concreta, ma al contempo di alto livello, dei dati. La soluzione infatti prevede di introdurre un modello di dati che funga da ponte tra i concetti dell'ontologia e i dati reali da analizzare, denominato *RADAR Data Model*.

Si tratta di un modello orientato agli oggetti, poiché il *RADAR Data Model* fornisce una visione concreta dei concetti nell'ontologia, in cui classi sono denominate *Classi concrete*. Gli esperti di dominio sono dotati di uno schema *RADAR Schema*, che rappresenta un'istanza specifica del *RADAR Data Model*.

Lo schema *RADAR Schema* deve essere mappato nello schema attuale dei dati sorgente. In questo modo, sarà possibile fornire agli analisti una visione di alto livello dei dati reali. Chiaramente, non tutte le proprietà definite nelle classi concrete sono necessariamente mappate in alcuni attributi del dataset sorgente. Dettagli riguardanti la definizione delle proprietà di una classe sono riportati in Sezione **??**.

Allo stesso modo il progetto realizzato prevede inoltre l'introduzione di un linguaggio per definire le regole di aggiornamento definite *Update Rules*, chiamato *RADAR Rules Language*, al fine di completare proprietà che non corrispondono agli attributi dei dati sorgente.

Il linguaggio delle regole è definito con l'obiettivo di fornire un modo non procedurale (ovvero non basato su tecniche standard) per definire i valori delle proprietà derivate, cioè le proprietà che non sono mappate su alcun attributo nelle tabelle sorgente. Le regole*RADAR* sono regole di tipo *Condizione-Azione* (CA) [20]: la loro azione viene eseguita quando la condizione è verificata; l'obiettivo della regola è la definizione di un'istanza di classe. Le regole sono associate ad una classe concreta definita all'interno di *RADAR Schema* e sono valutate quando una nuova istanza viene memorizzata nel database. La condizione di una regola viene valutata sui valori delle sue proprietà e, se vera, l'azione viene eseguita su di essa, per assegnare valori alle proprietà (si ricordi che le proprietà che non sono mappate in nessun attributo all'interno delle tabelle sorgente non hanno un valore. E' pertanto possibile definire nuove regole per assegnare un valore a tali attributi).

Le Relazioni concrete definite all'interno di *RADAR Schema* possono essere seguite, per ottenere o modificare i valori delle proprietà nella classi in relazione alla classe di applicazione. Come esempio, si consideri lo schema riportato in Figura 2. Si consideri la seguente *RADAR*

Rule.

```
Class:
 Financing_State
Condition:
 RATING 34 != 10 AND
 look-up(Cession via Related_Cession).PERFORMANCE
  != "Performing"
Action:
(Cessione via Relazione_Cessione).PERFORMANCE = "Delinquent"
```

Alla fine del caricamento di tutte le nuove istanze di Stato Finanziamento nel Database viene attivato il processo di calcolo delle regole. La condizione della regola controlla se il valore della proprietà `RATING34 !=10` è `!=10` e lo stato attuale della relativa sessione è `"Performming"`. L'operatore di `look-up`, descritto in Sezione 3.3, permette di raggiungere l'istanza della classe `Cession` seguendo la relazione di ricerca `Related_Cession`, vedere Figura 2. Se la condizione è vera, la proprietà `PERFORMANCE` è settata a `"Delinquent"`.

Lo scopo di queste regole è comunque quello di definire le indicazioni con cui ricavare le proprietà mancanti sui nuovi dati. Maggiori dettagli sono riportati in Sezione **??**.

Da un punto di vista generale, la preparazione dei template da inviare alle varie istituzioni si divide principalmente nelle seguenti fasi:

1. Definizione dei concetti: si definiscono i concetti (classi) con le relative proprietà che verranno successivamente impiegati per la generazione del report. Partendo dai concetti definiti nello Schema.org, si definiscono le classi dei concetti presenti, adattandole al contesto considerato.

2. Definizione delle relazioni: si definiscono le relazioni tra le classi.

3. Definizione delle regole: le regole vengono definite specificando le condizioni/azioni di calcolo delle proprietà presenti nelle classi.

4. Mapping delle proprietà: si mette in relazione il modello concettuale (le classi) con il modello logico (il Database).

5. Definizione del report: corrisponde alla scrittura del template del report da generare esplicitando la metodologia di calcolo delle aggregazioni.

6. Generazione delle query: generazione dell query SQL delle regole e delle aggregazioni definite precedentemente.

7. Compilazione del report: corrisponde all'esecuzione delle query sopra definite.

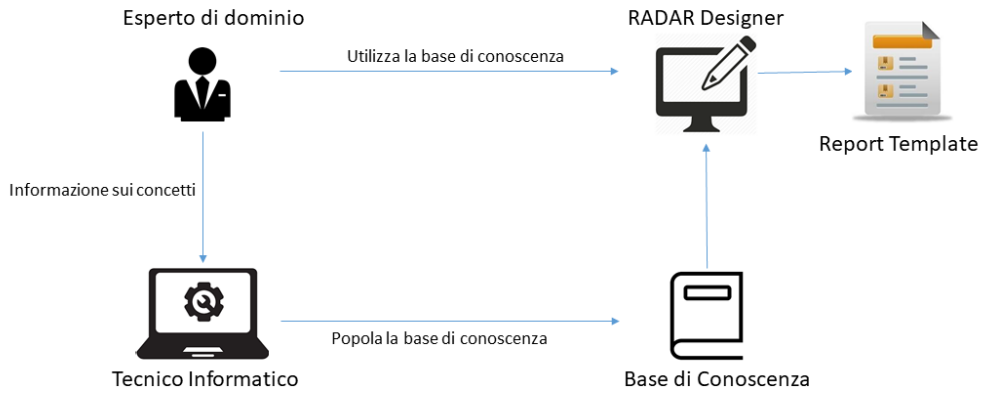

Figure 1: Visione del contesto

3.3 Struttura generale dei dati

La logica del modello dei dati definito nel presente progetto è quella di fornire agli esperti di dominio una visione dei dati che non sia di livello troppo elevato, al fine di poter essere concretamente utilizzata per specificare interrogazioni (query) ed aggregazioni. A tal proposito, è possibile affermare che quello implementato è un *modello logico di alto livello*, in quanto il suo livello è più alto delle tabelle relazionali, pur rimanendo tuttavia logico.

Il suo collegamento con i concetti dell'ontologia è essenziale per poter fornire agli utenti una visione concettuale in grado di supportare il processo di individuazione dei dati, al fine di estrarre le informazioni desiderate: l'ontologia è utile infatti nella ricerca e comprensione dei dati. D'altro lato le classi concrete permettono di specificare query ed aggregazioni. Lo schema *RADAR Schema* svolge quindi il ruolo di un'ontologia cosiddetta *ontologia applicativa*, che rende concreto il riferimento alla *Ontologia del dominio* (Schema.org, nel progetto).

I componenti proncipali vengono di seguito riportati.

- *Classi Concrete*: I dati sono descritti da una classe concreta. Tale classe deriva da una classe concettuale definita nell'ontologia di riferimento, dando una visione concreta di essa (ereditando cioè le proprietà dalla classe concettuale ed eventualmente definendo nuove proprietà). I dati effettivi sono *istanze* della classe (una classe concreta ha associato un contenitore di istanza). In particolare, nell'esempio riportato in Figura 2 sono definite tre classi radice concrete, ovvero Cessione, Stato Finanziamento e Piano Ammortamento. Le classi ereditano sia le proprietà della classe concettuale *Loan o Credit*, definita in Schema.org, sia le proprietà a loro volta ereditate dalla classe concettuale.

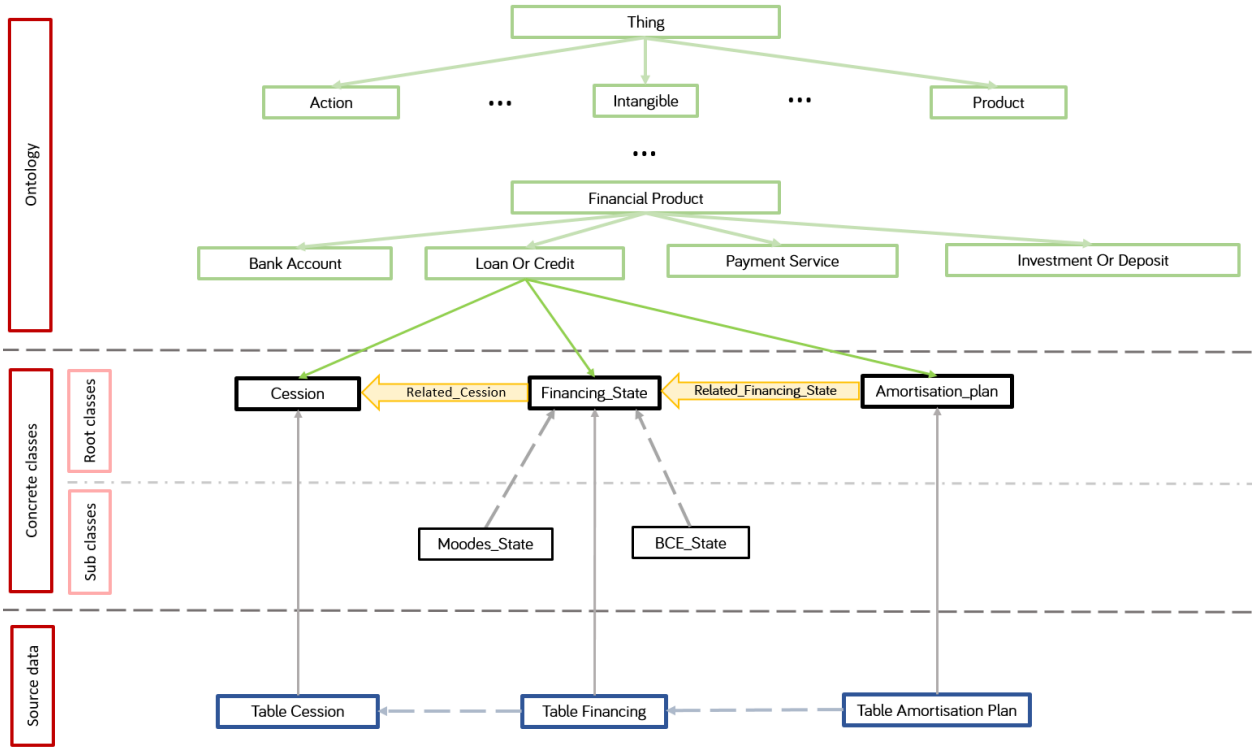

Figure 2: Rappresentazione grafica della vista unificata dell'ontologia (in alto), del modello logico di alto livello (al centro) e delle tabelle sorgente (in basso).

- *Sotto-classi Concrete*: Le classi concrete possono avere sottoclassi; una sottoclasse può essere ulteriormente specializzata in altre sottoclassi, e così di seguito. Una sottoclasse eredita tutte le proprietà dalla classe madre e può avere proprietà specifiche. Per esempio, in Figura 2 la classe concreta Stato Finanziamento è dettagliata in due sottoclassi, Sottoclasse1 e Sottoclasse2.

- *Relazioni di Look-Up*: Tra le classi radice concrete, è possibile definire relazioni di ricerca, indicate come $c_1 \rightarrow c_2$. Una relazione di ricerca denota che un'istanza di classe c_1 si riferisce ad un'istanza di classe c_2. La presenza di una relazione di ricerca denota l'esistenza di una relazione referenziale (Foreign Key) all'interno dei dati, anche se, in questo contesto, l'integrità referenziale (Reference Key) non è descritta nel DDL (Data Definition Language) del Database, in quanto assicurata dal processo di caricamento dei dati.

  In Figura 2 viene riportata ad esempio una relazione di ricerca (*Related_ Cession*) dalla classe concreta Stato Finanziamento (*Financing_ State*) alla classe concreta Cessione (*Cession*), in quanto uno stato di finanziamento si riferisce alla cessione che descrive.

  Riprendendo l'esempio precedente, una relazione è descritta nel seguente modo:

  ```
  look_up(Cession via Related_Cession)
  ```

- *Proprietà*: una classe possiede differenti proprietà, identificate da un nome univoco e con un tipo di dati specifico. Una proprietà viene classificata come *proprietà categorica* o *proprietà misura*. Una proprietà categorica si divide in *Categorico* (insieme di elementi non limitato) e *Categorico Finito* (insieme di elementi limitato). Le proprietà categoriche giocheranno il ruolo di *proprietà di raggruppamento* alla progettazione dei report. Una proprietà misura, di solito valore numerico, esprime il valore assunto da una proprietà.Esse verranno aggregate in modo da fornire dati aggregati nelle relazioni.

- *Chiavi*: Una classe radice concreta possiede una chiave, cioè un sottoinsieme non vuoto di proprietà, il cui valore identifica in modo univoco un'istanza della classe. Di conseguenza, quando viene definita una relazione di ricerca $c_1 \rightarrow c_2$, è necessario specificare l'elenco delle proprietà della classe c_1, che deve corrispondere alle proprietà della chiave della classe c_2.

Il livello superiore della Figura è rappresentato dall'*Ontologia*. Si noti che le tre classi concrete Cessione, Stato Finanziamento e Piano Ammortamento derivano tutte dalla stessa classe ontologica: *Loan or Credit*. Il concetto di *Loan* riportato nell'ontologia è molto generico e presenta diversi significati: esso infatti identifica il contratto (modellato dalla classe Cessione), lo stato attuale del prestito (modellato dalla classe Stato Finanziamento, e, le rate da pagare (modellate dalla classe Piano Ammortamento). Come sopra riportato è possibile quindi dedurre come una classe concreta sia definita tale in quanto è in grado di dare una visione concreta di un concetto generico nell'ontologia.

Il livello inferiore è rappresentato dai *Dati Sorgente*: i rettangoli corrispondono alle tabelle del database di origine. Le frecce che collegano una classe concreta ad una tabella denotano che l'attributo è mappato nella classe concreta. Questo significa che non necessariamente tutte le proprietà della classe avranno un valore proveniente dalla tabella.

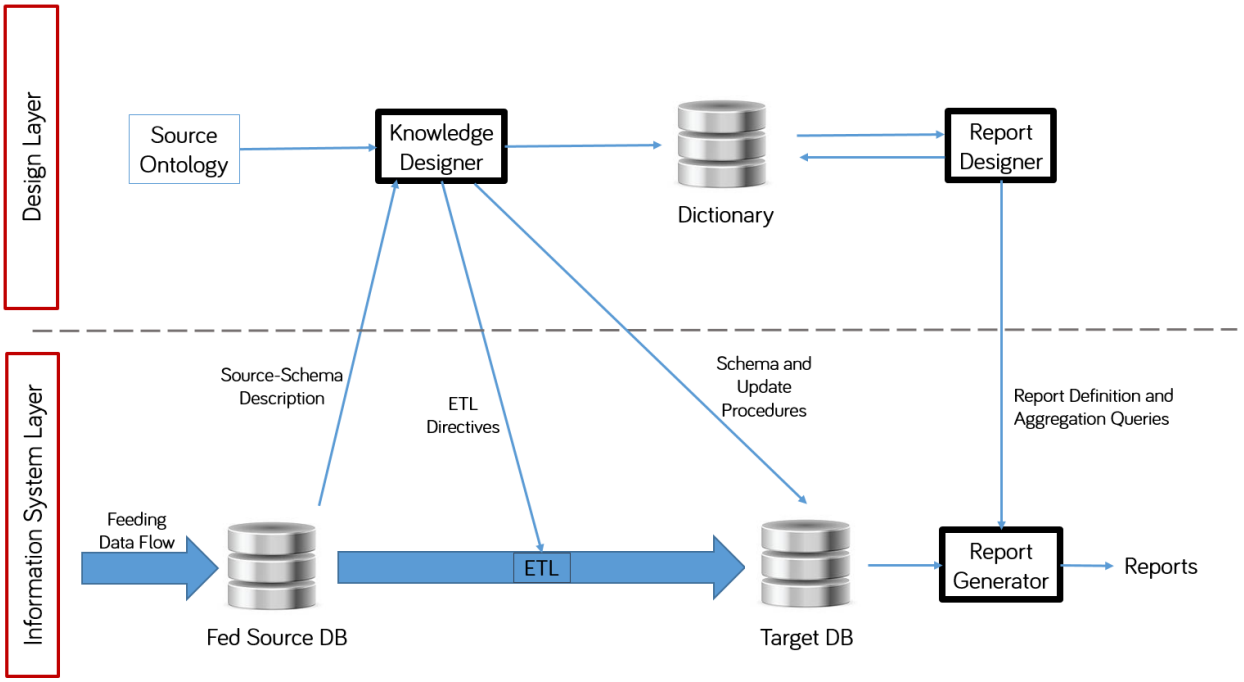

Figure 3: Architettura generale del Framework RADAR.

4 RADAR Framework

Come precedentemente anticipato, *RADAR* è un framework composto da vari strumenti, alcuni dei quali integrati nel sistema informativo che elabora i flussi dei dati. Di seguito vengono presentati l'architettura e il componente Knowledge Designer.

4.1 Architettura

La Figura 3 illustra l'architettura di *RADAR*. La linea tratteggiata separa i due livelli: sotto la linea tratteggiata, si trova l'elemento *Livello Sistema Informativo* (Information System Layer); sopra la linea tratteggiata, troviamo il *Livello Progettazione* (Design Layer). Di seguito, descriviamo in dettaglio il ruolo svolto dai due livelli.

Information System Layer : questo livello contiene i database, le applicazioni, i servizi e ogni risorsa computazionale che costituisce il sistema informativo. Per quanto riguarda il framework *RADAR*, questo livello presenta i servizi di seguito riportati.

- *Fed Source DB* corrisponde ad un qualsiasi tipo di sistema di archiviazione (ad esempio un database relazionale) che memorizza il *Feeding Data Flow*, ovvero quei dati che periodicamente vengono forniti in ingresso al sistema informativo.
- *Target DB* corrisponde al database (relazionale) in cui devono essere trasferiti i dati da analizzare, al fine di eseguire ulteriori aggregazioni che fungono da input per generare i reports. Processi *ETL* periodici (*Extract-Transform-Load*) dovrebbero eseguire questo

trasferimento. Questo database può essere visto come un *Operational Data Store* che fornisce una vista unificata di tutti gli insiemi di dati che lo alimentano.

- *Report Generator* ha lo scopo di generare effettivamente dei report, sulla base di una definizione di report precedentemente implementata, che si basa sulle query di aggregazione necessarie per calcolare le misure aggregate da inserire nei nuovi report generati.

Design Layer : questo layer gestisce la fase di progettazione del report e della conoscenza. Esso è associato al modulo *Information System Layer*, al fine di superare le restrizioni impostate dagli amministratori di sistema nell'implementazione di nuovi servizi all'interno del sistema informativo stesso. I principali componenti sono di seguito riportati.

- *Dictionary* è il database proprio del framework *RADAR*. Il nome è motivato dal fatto che il framework *RADAR* gestisce un'ontologia, lo schema *RADAR Schema*, e aggiorna le regole che descrivono tutti i dati e i termini corrispondenti.

- *Knowledge Designer* è il componente del framework *RADAR* che fornisce l'interfaccia utente dello *RADAR Schema* (memorizzato nel *Dictionary*).

- *Ontologia Sorgente* (o *Reference Ontology*) viene caricato dal *Knowledge Designer* e salvato nel *Dictionary*, in modo da costituire la base per definire lo *RADAR Schema*.

- *Report Designer*, è la seconda interfaccia utente del framework *RADAR*. E' progettato per fornire agli analisti un'interfaccia user-friendly, in grado di sfogliare la base di conoscenza finora costruita (ontologia e *RADAR schema*), definire aggregazioni e progettare report.

Relazioni fra i Layers I componenti dei layers sopra descritti sono messi in relazione fra loro attraverso una serie di operazioni di seguito riportate.

- *Descrizione dello Schema Sorgente* è ricevuto dall'*Information System Layer* e descrive descrive i set di dati memorizzati nel *Fed Source DB*. Questa descrizione è essenziale per far sapere al progettista dello *RADAR Schema* quali dati (tabelle e campi) sono effettivamente disponibili nel *Fed Source DB*; inoltre, è essenziale per guidare ulteriormente le attività *ETL* dal *Fed Source DB* al *Target DB*.

- Una volta che lo *Schema RADAR* è definito "stabile", esso deve essere tradotto nello schema relazionale da distribuire nel *DataBase Target*. Lo schema è seguito da cosiddette *procedure di aggiornamento*, che alimentano il *DataBase Target* attraverso procedure di aggiornamento generate dalle cosiddette *RADAR Rules*. Tali procedure vengono eseguite al termine del processo di *ETL* sui nuovi dati aggiunti al *Target DB*, in modo da rimuovere l'incompletezza dei dati ed, eventualmente, aggiornare i dati già memorizzati nel *Target DB*.

- *Direttive ETL* sono generate al fine di consentire al servizio *ETL* di trasferire i datasets dal *Fed Source DB* al *Target DB*.

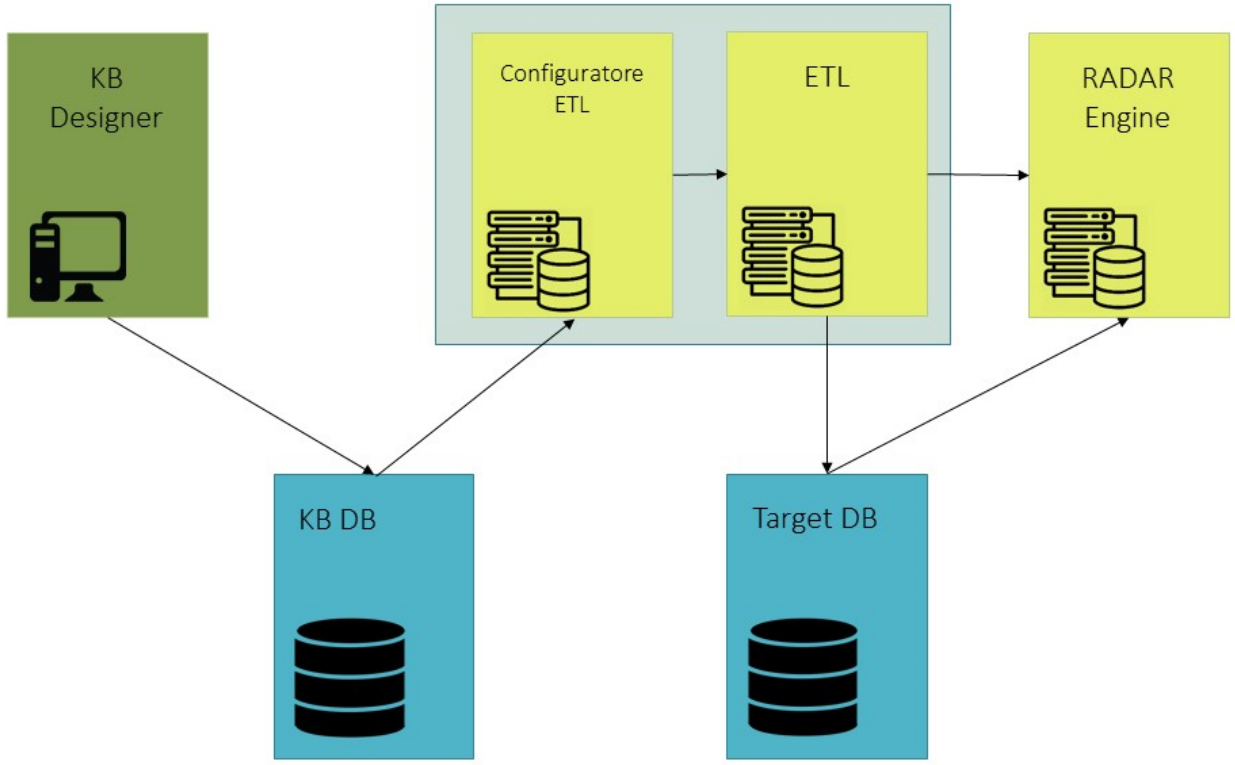

Figure 4: Rappresentazione dei componenti Hardware del Framework RADAR.

- *Definizioni del Report* e *Queries di Aggregazione* sono sviluppati nel *Report Generator* (nel *Information System Layer*), per renderlo effettivamente in grado di generare report.

4.2 Architettura Hardware

L'architettura Hardware del Framework RADAR è descritta in Figura 4.

In particolare l'eseguibile del Knowledge Base (KB) Designer viene lanciato su una macchina che acquisisce informazioni dal DataBase del Knowledge Base, ovvero dal KB DB, che a sua volta è in comunicazione con il Configuratore dell'ETL, che viene eseguito su un server a parte. L'eseguibile dell'ETL può essere successivamente installato e lanciato sullo stesso server del configuratore, o su un server a parte. L'ETL è collegato direttamente al Target DB, DB a cui attinge informazioni anche il RADAR Engine, il cui eseguibile è lanciato su un server a parte.

5 Knowledge Designer in Dettaglio

Il Knowledge Designer è il componente del framework RADAR che viene utilizzato al fine di costruire lo Schema *RADAR Schema* e l'intera base di conoscenza memorizzata all'interno del dizionario. Di seguito vengono riportate le attività eseguite all'interno del componente.

1. La prima attività da eseguire è quella di caricare la *Source Ontology*, che meglio si adatta al contesto applicativo. In questo esempio corrente riguardante i prestiti, abbiamo adottato l'ontologia *Schema.org*, perché è adatta a fornire i concetti generici riguardanti i prodotti finanziari.

2. Viene definito lo *RADAR Schema* per lo specifico contesto applicativo. La figura **??** riporta uno screenshot dell'interfaccia utente per progettare classi concrete. Sul lato sinistro della figura, è possibile notare il browser *Ontology Browser*, che permette di selezionare la classe ontologica (concetto) da cui deriva la nuova classe concreta. Una volta selezionata la classe ontologica, viene mostrata una finestra che fornisce gli strumenti per definire lo schema della nuova classe concreta. In Figura **??**, viene mostrata questa finestra per definire la classe concreta *Cession*, il cui schema è riportato in Tabella **??**. Si noti che le proprietà ereditate non appaiono, perché sono mostrate in una fase preliminare, una volta selezionata la classe ontologica.

3. Lo schema *RADAR Schema* è completato dalla definizione di relazioni di ricerca. La Figura **??** mostra uno screenshot della finestra per definire le relazioni di ricerca. In particolare, viene riportata la definizione della relazione *Related-Cession*, presentata e mostrata in Figura 2. In particolare, nella classe di riferimento (in questo caso, la classe *Finanziamento_Stato*, a sinistra) sono associate a nella chiave della classe di riferimento (in questo caso, classe *Cession*, a destra).

4. *Definizioni dello Schema* sono ricevute dall'applicazione *Fed Source DB*, per conoscere le attuali strutture dei dati da associare alle classi concrete.

5. Un'attività di *Mapping* è necessaria per associare classi concrete con i dataset presenti nel Fed Source DB: attraverso questa attività è possibile sapere esattamente quali proprietà (attributi) sono disponibili nel DB sorgente e quali devono essere derivati.

6. Successivamente all'attività di *Mapping*, è necessaria la definizione delle regole (si veda Sezione **??**) del framework per il calcolo delle proprietà derivate, cioè le proprietà che non sono mappate su nessun attributo delle tabelle sorgenti.

7. Questa attività genera lo Schema e le procedure cosiddette di "Update", necessarie alla preparazione del DB Target, come anche le direttive ETL, al fine di generare ed eseguire i processi ETL che effettuano il trasferimento dei dati dal Fed Source DB al DB Target.

Il Knowledge Base Designer ha il compito di gestire i vari dizionari, le rispettive definizioni, la modifica e la loro consultazione. Seguendo l'approccio descritto in Schema.org Evolution of Structured Data on the Web, il Browsing dei concetti, ma più in generale il Knowledge Base Designer si sofferma sulla descrizione del layer di presentazione, mentre sarà compito del RADAR Engine aggiungere il layer dei contenuti, cioè i dati. Si divide in varie parti, ognuna con un compito specifico:

- Classes Designer
- Relation Classes Designer

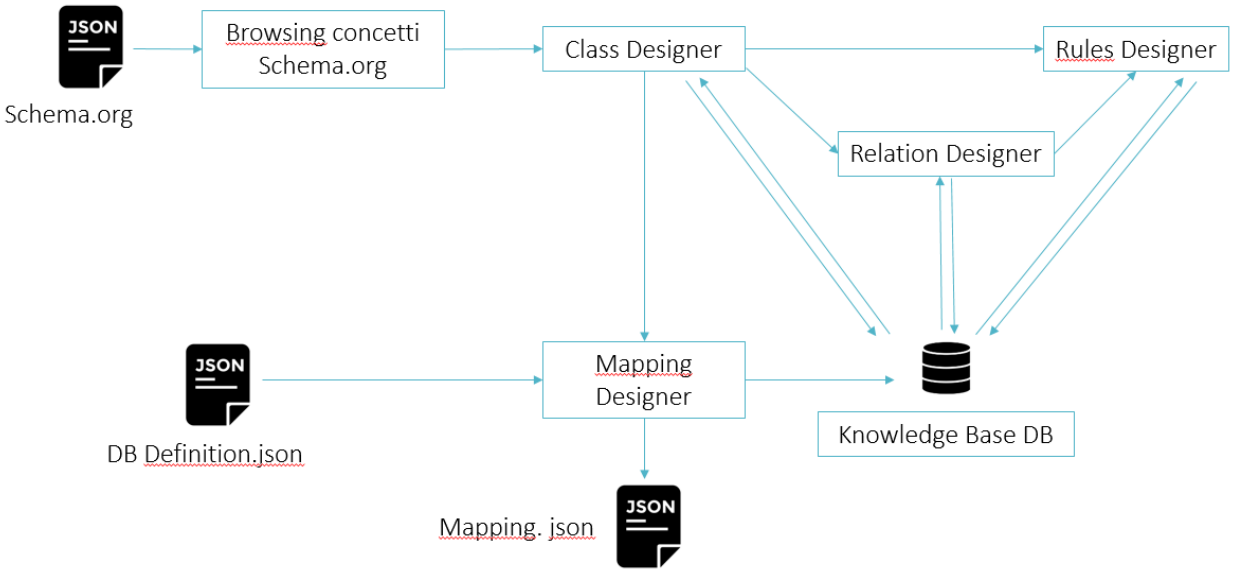

Figure 5: Architettura del Knowledge Base Designer.

- Mapping Designer
- Rule Designer
- Context Parameters Designer
- Knowledge Base DB

La Figura 5 ne riporta il flusso operativo. Partendo dalla navigazione del browsing dei concetti dello Schema.org si definiscono i concetti e attraverso l'Interpratation Designer si specificano le varianti dei vari concetti.

In parallelo in questa potrebbe già partire la fase di mapping operata attraverso il mapping designer.

Dopo aver definite le varie interpretazioni dei concetti, si definisco le regole di calcolo delle proprietà e si generano le regole.

A questo punto può iniziare la fase di mapping, prendendo in input la struttura dei dati finanziari (nel nostro caso file excel), e creare il collegamento tra struttura concettuale e logica dei dati (Mapping Designer). Per semplificare il lavoro dell'umano la fase di mapping è preceduta da una fase di Table Analyzer, dove si cerca di automatizzare la fase di riconoscimento delle proprietà e misure.

5.1 Classes Designer

Il ruolo del Classes Designer è quello di definire una nuova classe all'interno dell'ontologia. La definizione di una nuova classe inizia attraverso il Browsing Schema.org Concept, ovvero una GUI che visualizza i contenuti dell'ontologia (Schema.org) e relative proprietà, visibile in Figura **??**. Partendo dal concetto base, scelto al passo precedente (Browsing Schema.org

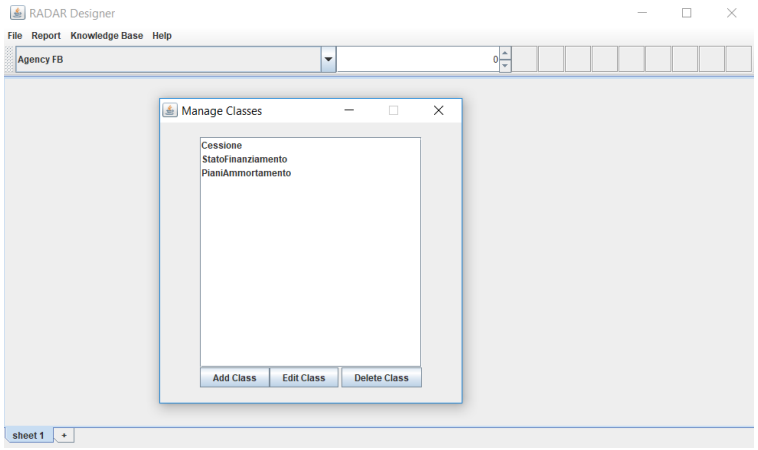

Figure 6: Form di Gestione delle Classi

concept) e contenuto nello Schema.org, si definiscono le varie classi, ereditando alcune proprietà già presenti nella superclasse scelta, ed aggiungendone altre non presenti. Un classe per RADAR framework necessita della definizione delle proprietà chiave. La definizione delle proprietà di una classe può avvenire in due modi, ereditando proprietà già presenti nelle classi padre, o definendo una nuova proprietà attraverso la form visibile in Figura 10.

Per quanto riguarda il settore finanziario Schema.org descrive le classi FinancialProduct e LoanOrCredit, partendo da queste superclassi che definiscono concetti generali si possono definire le classi, per esempio: stato finanziamento, piano d'ammortamento e cessione.

Il framework RADAR permette l'aggiunta, la modifica e la rimozione di classi e delle corrispondenti proprietà, come mostrato negli screenshot di seguito riportati. Una classe può essere aggiunta attraverso la form riportata in Figura 6.

In particolare, La Figura 7 mostra l'attività di Browsing dei concetti presenti in Schema.org, dove ad ogni concetto è associata la corrispondente definizione testuale, mentre la Figura 8 evidenzia le proprietà definite all'interno della classe aggiunta, selezionata dal browsing riportato nella figura precedente.

Le proprietà di una Classe sono gestite attraverso una form come riportato in Figura 10. In particolare, per ciascuna proprietà vengono specificate informazioni quali il nome, il tipo e la descrizione. Viene inoltre specificato se la proprietà è chiave, e il tipo di attributo, che può essere categorico (enumerativo 'aperto', ovvero senza attributi prefissati), categorico finito (enumerativo con valori ben precisi), o misura (valore numerico che rappresenta il valore assunto da una misura). E' possibile aggiungere, modificare od eliminare una proprietà di una classe sempre attraverso la form riportata nella suddetta figura.

Considerazione classes Designer

- Devo tenere in considerazione i duplicati (ciò le proprietà da me definite rispetto alle nuove proprietà divenute uno standard di Schema.org).

  Problema: Non devo ragionare solo per proprietà, ma anche interpretazioni, in quanto una interpretazione potrebbe essere una nuova classe di Schema.org.

  Gestire il problema delle sottoclassi, potrebbe essere più conveniente che una sottoclasse sia sottoclasse di una nuova sottoclasse

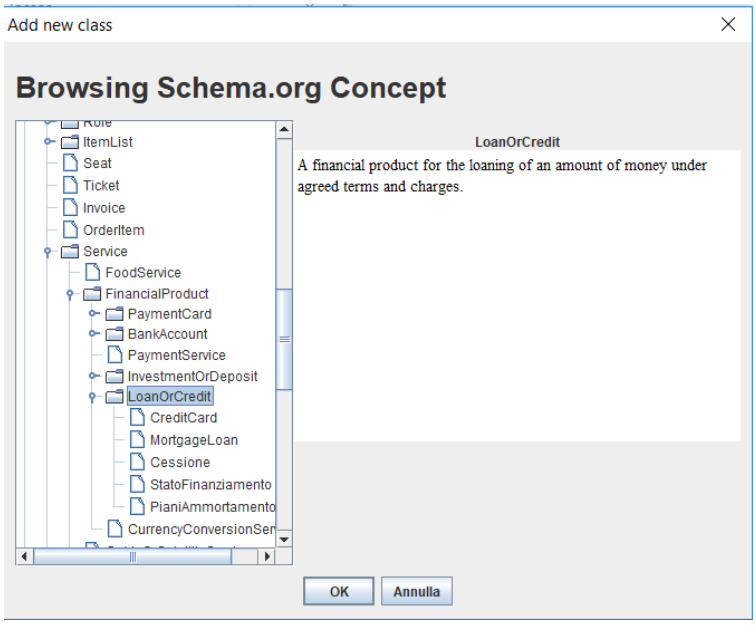

Figure 7: Form rappresentante l'attività di Browsing dei concetti di Schema.org.

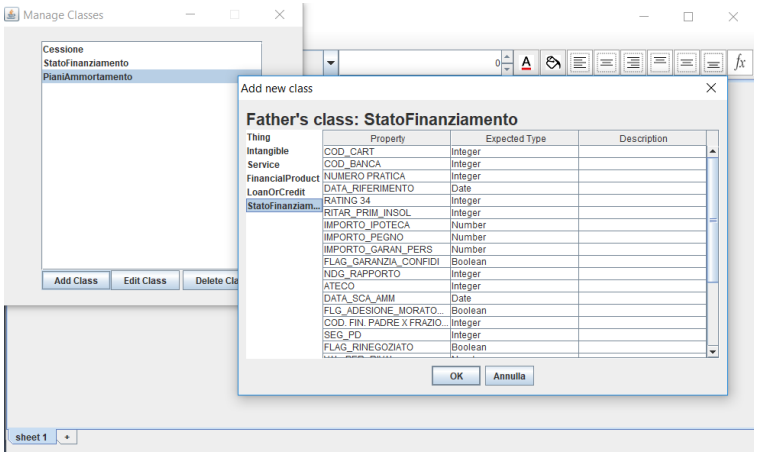

Figure 8: Form di aggiunta di una classe e delle corrispondenti proprietà.

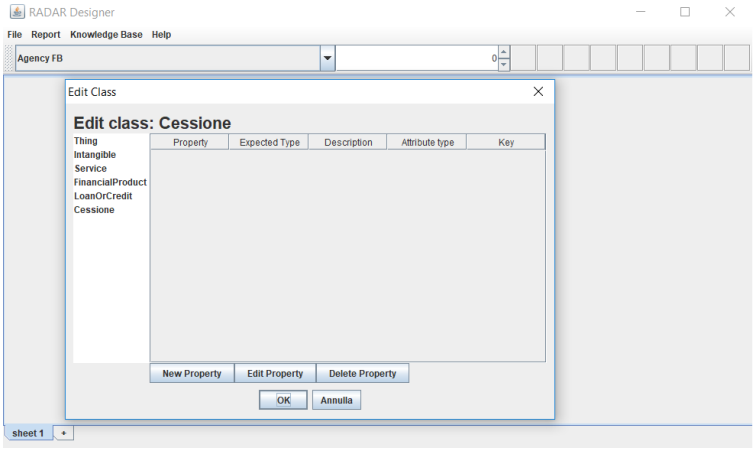

Figure 9: Form di Editing delle Classi

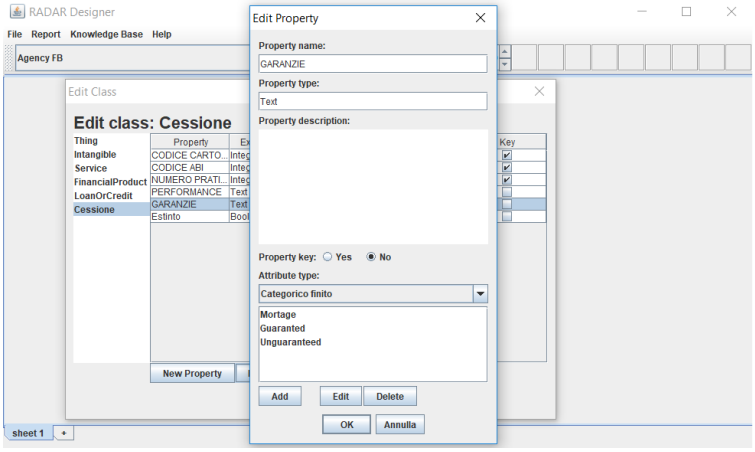

Figure 10: Form di Editing delle Proprietà di una Classe.

Possibile soluzione: Lavorare parallelamente con una doppia etichetta! Non risolve il problema delle interpretazioni (potrebbe essere necessario creare un nuovo livello, oppure dividere una interpretazione.

SOLUZIONE: Proponiamo noi una nostra estensione di Schema.org

- Come considero il problema di etichettatura dello storico in caso di cambio delle etichette e di introduzione di una nuova versione dello Schema.org

Schema.org, ontologia che descrive argomenti di carattere generale, nel caso si abbia la neccessità di avere una ontologia più specifica in un determinato ambito sufficente adattare la classe java di lettura del file dell'ontologia, oppure se può decidere anche di creare una ontologia "proprietarià" (crei da zero).

5.2 Relation Classes Designer

Le Classi definite nel modulo "Classes Designer" possono essere in relazione fra loro, e tali relazioni sono definibili attraverso la form rappresentata in Figura 11. E' possibile attraverso la form aggiungere, eliminare o modificare una relazione fra classi. In quest'ultimo caso la form rappresentata in Figura 12 mostra il dettagli con cui i componenti delle classi entrano in relazione fra loro. ANTONIA: CORRETTA LA NUMERAZIONE DELLE FIGURE?

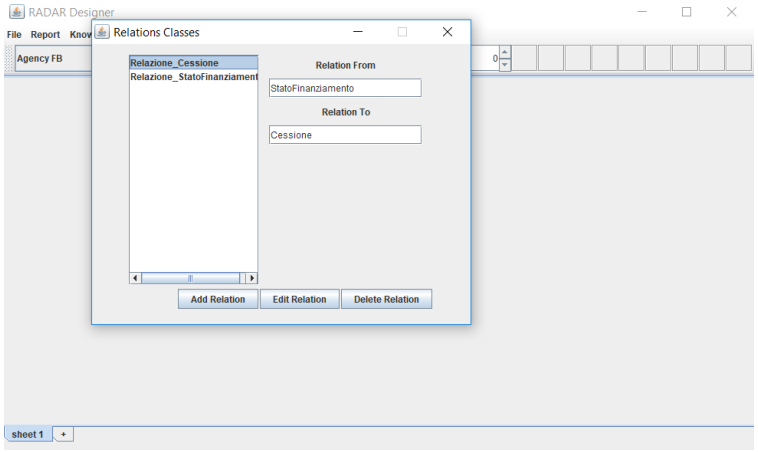

Figure 11: Form di gestione di una relazione fra le Classi.

5.3 Mapping Designer

Il modulo di Mapping Designer è definito all'interno dell'architettura come uno dei moduli "centrali", il cui compito è quello di conettere la struttura logica del database (definita nel file Json DB Definition, che viene acquisito in input) con la reale struttura fisica della sorgente dati (il Knowledge Base DB). Lo strumento cerca di mettere in relazione le due sorgenti dati automatizzandone il processo ed analizzando i nomi dei campi. Nel caso in cui non esista una corrispondenza fra la struttura logica e quella fisica, la relazione verrà definita dall'esperto umano.

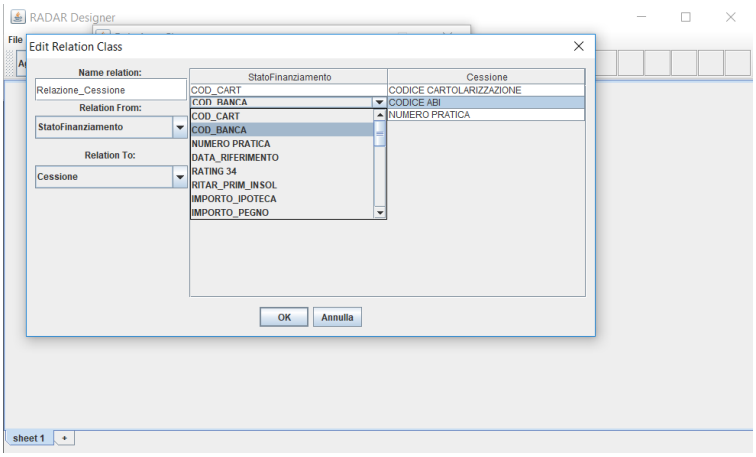

Figure 12: Form di modifica dei componenti di una relazione fra le Classi.

Il mapping è eseguito considerando proprietà di una classe, definendo una tabella definita dalle seguenti colonne (Figura 14):

- property: è la colonna contenente il nome delle proprietà della classe su cui si vuole effettuare il mapping.
- table attribute: contiene tutti gli attributi che saranno presenti nella tabella, scelta tra quelle del db target, su cui si vuole effettuare mapping.
- source attribute: header della sorgete dati (file CSV, tabella database sorgente, ...) su cui si vuole effettuare mapping.

Nella Mapping Collection definita nel Knowledge Base DB viene salvata per ogni proprietà della classe la coppia property - table attribute, creando un collegamento tra il dizionario e il target db.

Al contrario la seconda e la terza colonna non vengono salvate nel db, ma generano un file json contenente una lista di oggetti dove ogni oggetto è rappresentato dalla coppia table attribute - source attribute e la posizione del source attribute nel file CSV, ovvero il numero di colonna. Il file generato serve per produrre il file xml di mapping per l'ETL.

Il componente del Mapping Designer riceve quindi in ingresso i seguenti files di input:

- CSV file: contenente header degli attributi sorgenti (source attribute) e relative posizioni.
- Json file: contenente la struttura del target DB

Il file JSON contenente la descrizione del target DB presenta la seguente struttura:

```
[{
   "tableSource": "pianiAmmortamento",
   "attributes": [
        {
```

```
    "attributeName": "CODICE ABI",
    "expectedType": "BIGINT",
    "attributeType": "Categorico",
    "key": True
    },
{
"attributeName": "CODICE CARTOLARIZZAZIONE",
"expectedType": "BIGINT",
"attributeType": "Categorico",
"key": True
},
...
]},
{
"tableSource": "statoAmmortamento",
"attributes": [
{
"attributeName": "COD_BANCA",
"expectedType": "BIGINT",
"attributeType": "Categorico",
"key": True
},
{
"attributeName": "NUMERO PRATICA",
"expectedType": "BIGINT",
"attributeType": "Categorico",
"key": True
},
]}
    ]
```

Il file JSON prodotto presenta la seguente struttura:

```
    {
        "tableSource": "pianiAmmortamento",
        "mappingList": [
            {
              "csvCaption": "CODICE ABI",
              "csvPosition": 0,
              "targetName": "CODICE_ABI"
            },
            {
              "csvCaption": "CODICE CARTOLARIZZAZIONE",
              "csvPosition": 1,
              "targetName": "CODICE_CARTOLARIZZAZIONE"
```

```
        },
        {
          "csvCaption": "DATA RIFERIMENTO",
          "csvPosition": 2,
          "targetName": "DATE_RIFERIMENTO"
        },
        ...
    ]
}
```

Le form sottostanti (Figura 13) riportano i file con estensione .json e .csv che devono essere caricati e che corrispondono rispettivamente alla struttura del DataBase (BD) e alla struttura della sorgente dati.

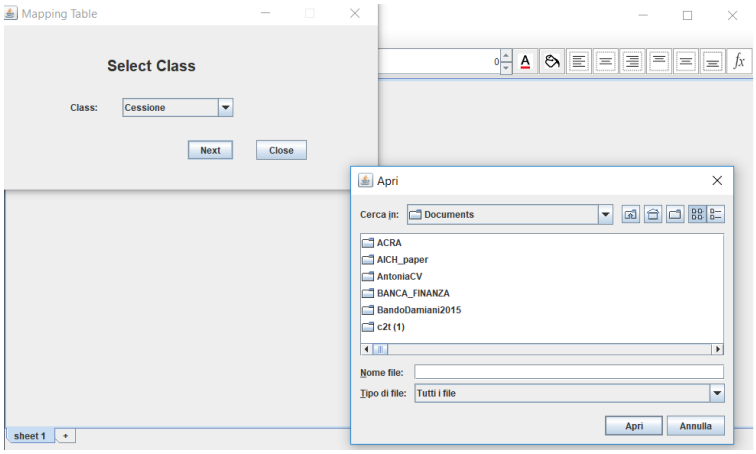

Figure 13: Form di inserimento dei file .json e .csv per il mapping.

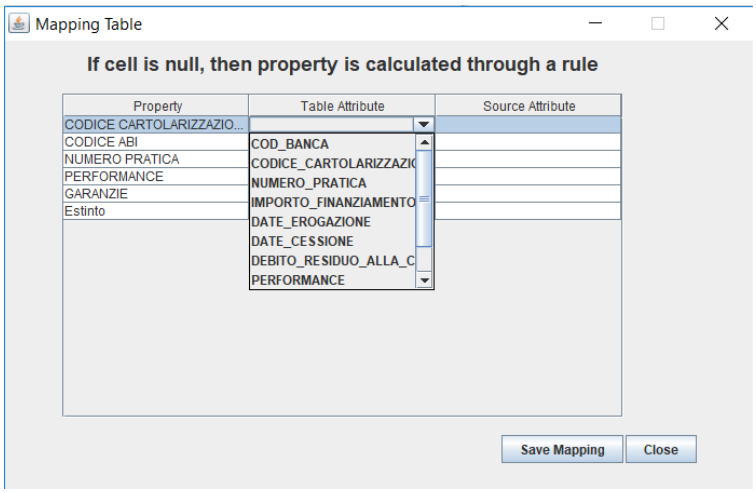

Figure 14: Tabella di mapping Cessione; settaggio delle proprietà.

Una volta caricate le due strutture dati, il framework eseguirà il mapping fra il DB, la struttura del dizionario e la struttura della sorgente dati.

5.4 Rule Designer

Il Rule Designer è rappresentato attraverso un'interfaccia grafica che supporta l'utente nella definizione delle regole delle propietà di una classe. L'input della regola avviene attraverso un editor grafico, dove l'utente, con una serie di menu a tendina, può costruire le proprie regole (condizioni) di calcolo delle proprietà e misure. Le figure di seguito riportate riassumono i vari passaggi eseguiti per la loro definizione.

In particolare, la Figura 15 mostra la possibilità di aggiungere, modificare od eliminare delle regole, siano esse le principali, associate ad una classe, oppure definite ad hoc (ovvero customizzate) per una determinata classe. Per ciascuna di esse viene riportata la priorità.

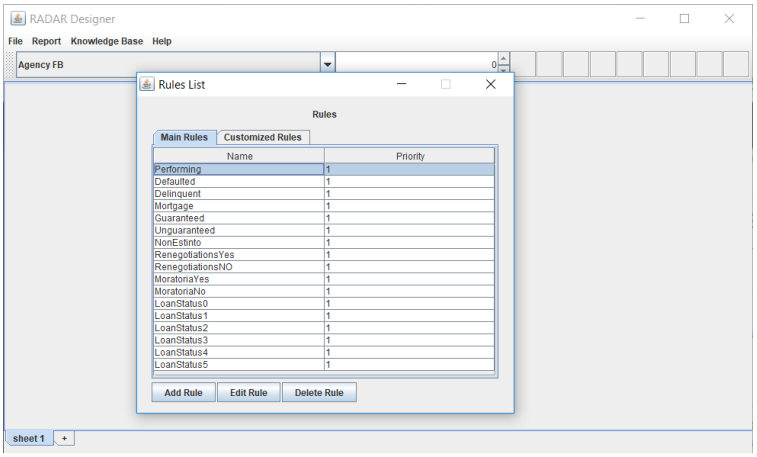

Figure 15: Form di visualizzazione delle regole principali.

Semplificazione Il calcolo della priorità delle regole è un'operazione complessa in quanto alcune regole necessitano del risultato di attributi che vengono a loro volta calcolati con altre regole. Per semplificare il riconoscimento della precedenza di calcolo delle regole, l'utente stabilirà la priorità di ogni regola. Le regole saranno eseguite con un ordine crescente di priorità.

Una regola customizzata, descrive una eccezione di calcolo di una regola principale, cioè ridefinisce la condizione / azione di calcolo della regola rispetto ad una properità quando essa assume determinati valori.

La Figura 16 mostra la form che permette all'utente di definire una nuova regola, defnire nuove condizioni e operatori logici corrispondenti, oltre che a modificarli. La form permette inoltre l'eliminazione di una condizione esistente. Oltre alla definizione della condizione (IF) di una regola, nella form viene inoltre specificata la conseguenza (THEN) ad essa associata, con la possibilità, attraverso l'opzione "Add new Comparison" di aggiungere ulteriori azioni conseguenti.

Gli argomenti degli operatori logici che definiscono le condizioni sono espressi attraverso la form riportata in Figura 17.

Come mostra la Figura 17, ciascun operatore logico può essere rappresentato attraverso una delle seguenti possibilità:

- una proprietà della classe,

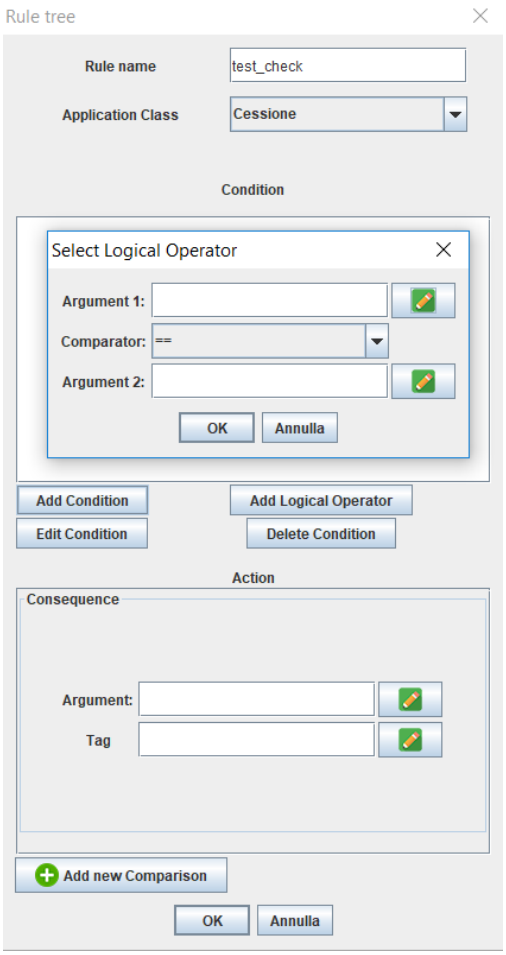

Figure 16: Form di aggiunta di una regola e delle corrispondenti condizioni.

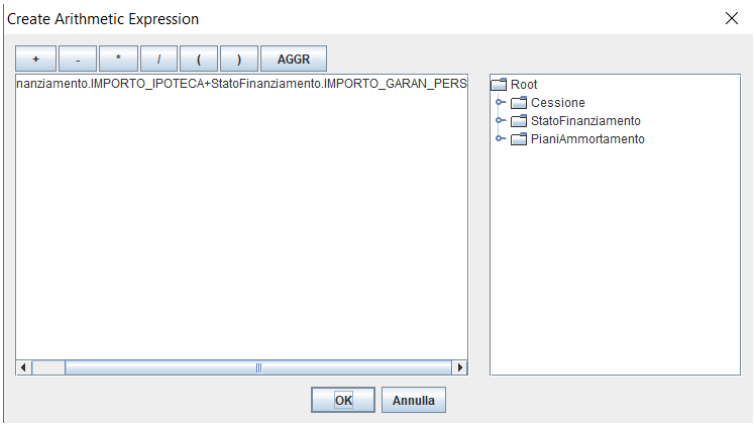

Figure 17: Form di scrittura di una espressione aritmetica.

- la risultante di un'espressione aritmetica,
- la risultante di un'aggregazione,
- la risultante di una combinazione matematica fra un'espressione aritmetica e una o più aggregazioni.

Le aggregazioni vengono eseguite attraverso l'utilizzo di un tool custom.

La LookUp Relation (relazione di lookup), che può essere inserita attraverso la Form di aggregazione 18, permette di eseguire un'azione in un'altra classe.

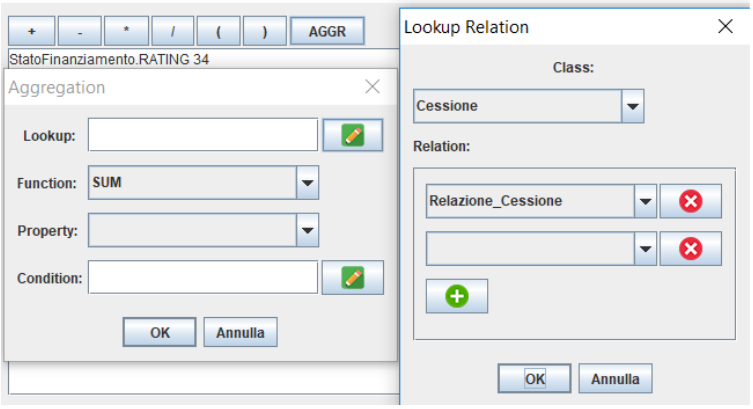

Figure 18: Form di aggregazione.

5.5 Context Parameters Designer

Il Context Parameter Designer definisce i parametri di contesto per il calcolo delle regole, cioè definiscono un sottoinsieme di dati contenuti nel DB target all'interno del quale vengono applicate le regole.

Di seguito viene riportatat la Form di definizione di un nuovo parametro, attraverso la quale si indica per ogni classe a quale proprietà si riferisces il nuovo parametro (es, parametro codice banca, in Cessione è Codice ABI, in Piano Ammortamento è codice ABI, in Stato Finanziamento è codice Banca), come riportato in Figura 19.

5.6 Knowledge Base DB

Come mostrato in Figura 5 il componente Knowledge Base DB è responsabile del salvataggio dei dati derivanti dai componenti del Knowledge base Designer. Il DBMS utilizzato in RADAR Framework per l'implementazione del Knowledge Base DB è MongoDB. Esso, come riportato anche nella letteratura [2], è uno dei più noti database non relazionali (o NoSQL). Si tratta di una soluzione orientata ai documenti, che sfrutta il formato JSON per la memorizzazione e la rappresentazione dei dati.

All'interno del DB sono presenti cinque Collection (Classes, Rules, Relations, Mapping e ContextParameter), una per ogni componente del KB Designer. Si noti che, poiché si tratta della definizione di un DB non relazionale, in Mongodb è possibile definire documenti

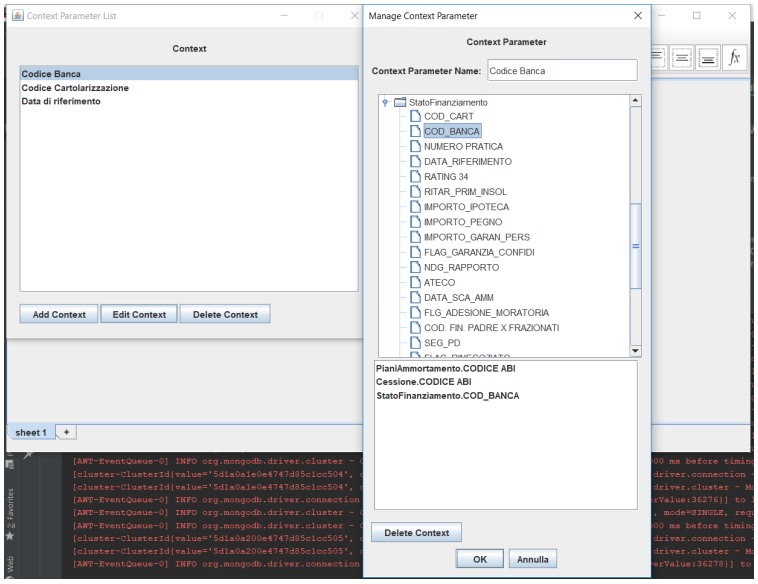

Figure 19: Form di definizione di un nuovo parametro.

(record di una collection) che presentano una strutta diversa per ogni documento presente nella collection. Un esempio è riportato all'interno della collection classes, dove l'attributo 'ValueAttributeType' assume valori validi solo per il caso in cui l'oggetto descritto sia di tipo 'categorico finito'.

Di seguito si riporta la struttura generale di ogni documento delle varie collection.

Classes Collection

- InterpretationName: nome della classe
- SubClassOf: rappresenta il percorso dell'albero con il browsing rispetto a schema.org (la radice dell'albero è indicata con 'thing').
- properties: Array of Objects: vettore delle proprietà di una classe
  - Property: nome della proprietà
  - Expected Type: tipo della proprietà
  - Description: descrizione
  - Attribute Type: tipologia dell'attributo
  - Key: specifica se è un attributo chiave
  - ofClass: specifica a quale classe fa riferimento
  - ValueAttributeType: attributo opzionale e presente solo per il categorico finito.

Relations Classes Collection

- name relation: nome della relazione
- class from: classe di partenza della relazione

- class to: classe di arrivo della relazione
- relation properties: vettore di coppie property from - property to
  - property from: proprietà da cui deriva la relazione
  - property to: chiave della classe di arrivo della relazione

Rules Collection

- rules name: nome della regola
- rule: condizione della regola
- priority: priorità della regola
- actions: vettore contente le azioni della regola
  - argument: proprietà da valorizzare al verificarsi della condizione
  - tag: valore che assume la proprietà al verificarsi della condizione

Mapping Collection

- interpretation: nome della classe da mappare
- mappingTable: nome della tabella nel db target da mappare
- mapping: vettore contentente le coppie mappate
  - property: nome della proprietà della classe
  - attribute: nome dell'attributo della tabella

Context Parameter

- name: nome del parametro di contesto
- properties: vettore con le proprietà di applicazione del parametro
  - name Property: nome della proprietà di applicazione del parametro

La Figura 20 riporta un estratto della struttura di una classe definita all'interno di Mongodb.

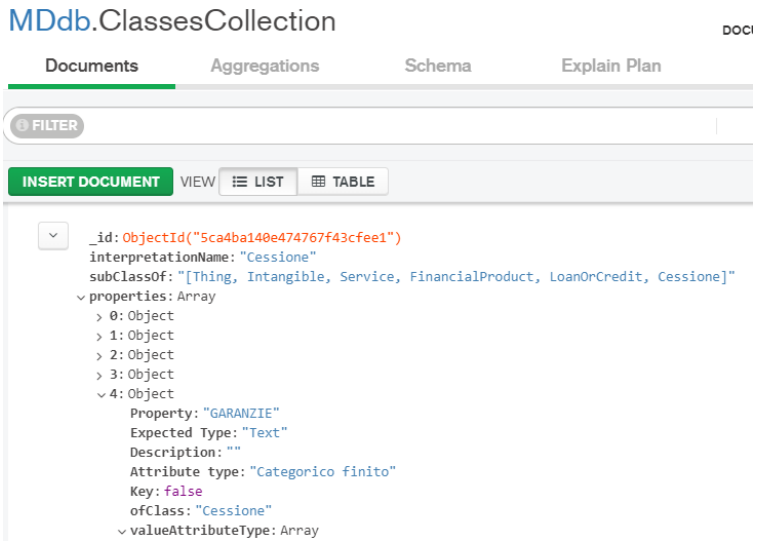

Figure 20: Rappresentazione grafica della struttura della classe Cessione e delle specifiche di uno degli oggetti in essa definiti.

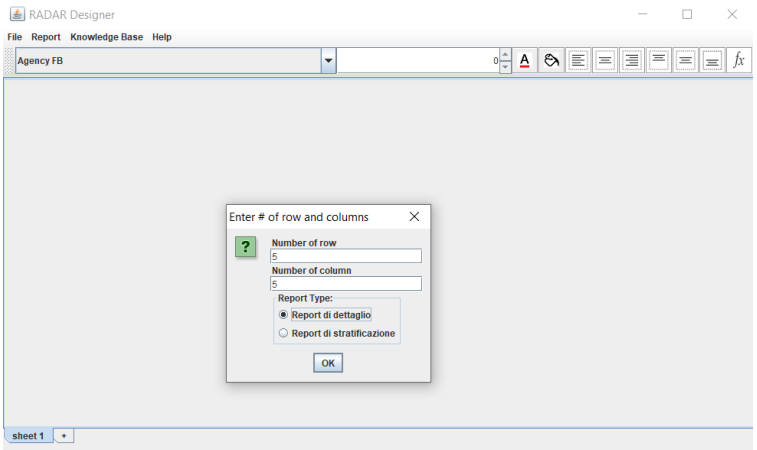

Figure 21: Form di defnizione della struttura (numero di righe e colonne) di una nuova tabella definita in un Foglio.

6 Report Designer

Il Report Designer (RD) è lo strumento grafico impiegato per la creazione del template del report. Dopo che l'esperto di dominio e l'esperto informatico hanno concluso la fase di *Knowledge Designer*, inizia la fase di creazione del report, cioè la definizione di tutte le query che sono neccessarie alla generazione del report stesso. Quest'ultimo è considerato un oggetto complesso, in quanto può essere costituito da più fogli, ciascuno dei quali può presentare più sezioni (ovvero tabelle). Per semplicità è stata definita la creazione di una sola tabella per ciascun foglio del report. Le tabelle vengono create dall'utente in ciascun foglio utilizzando la form di seguito riportata in Figura **??**.

La tabella così definita viene quindi visualizzata all'interno del foglio corrente, come riportato in Figura 22. Come si osserva, è possibile rinominare i fogli su cui vengono salvate le varie tabelle.

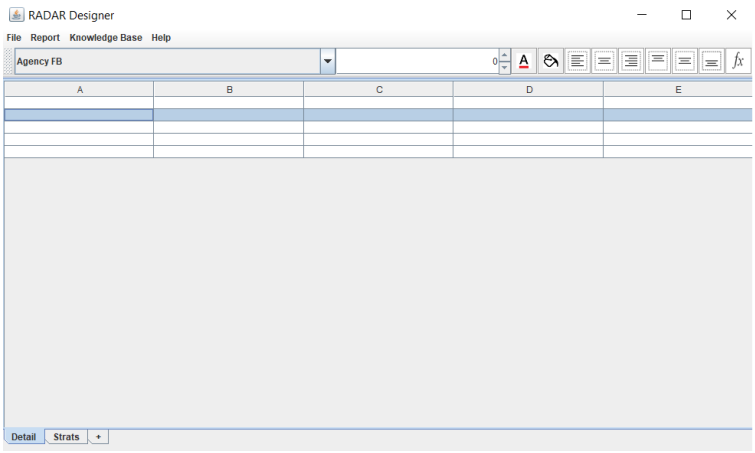

Figure 22: Form di visualizzazione di una nuova tabella definita in un Foglio.

Semplificazione Possibili sviluppi futuri del progetto potrebbero prevedere la possibilità di selezionare, attraverso una modifica, aree delimitate di un foglio a cui applicare determinate condizioni. In tal modo sarà possibile gestire più tabelle (definite in subfogli) all'interno di un unico foglio.

La creazione di un report avviene attraverso l'aggiunta di una tabella all'interno del foglio (menu report -> add table) e specificandone la tipologia di report che si vuole creare:

- Report di stratificazione
- Report di dettaglio

I fogli del report sono gestiti attraverso i comandi presenti nel footer del Report Designer. Ogni report presenta dei parametri di gestione (RP - Report Parameter). Essi vengono scelti per selezionare i dati che saranno utilizzati per la generazione del report. Ad esempio il parametro "Data di Riferimento" indica la generazione di un report utilizzando i dati presenti alla data di riferimento. La gestione dei parametri avviene attraverso form riportate

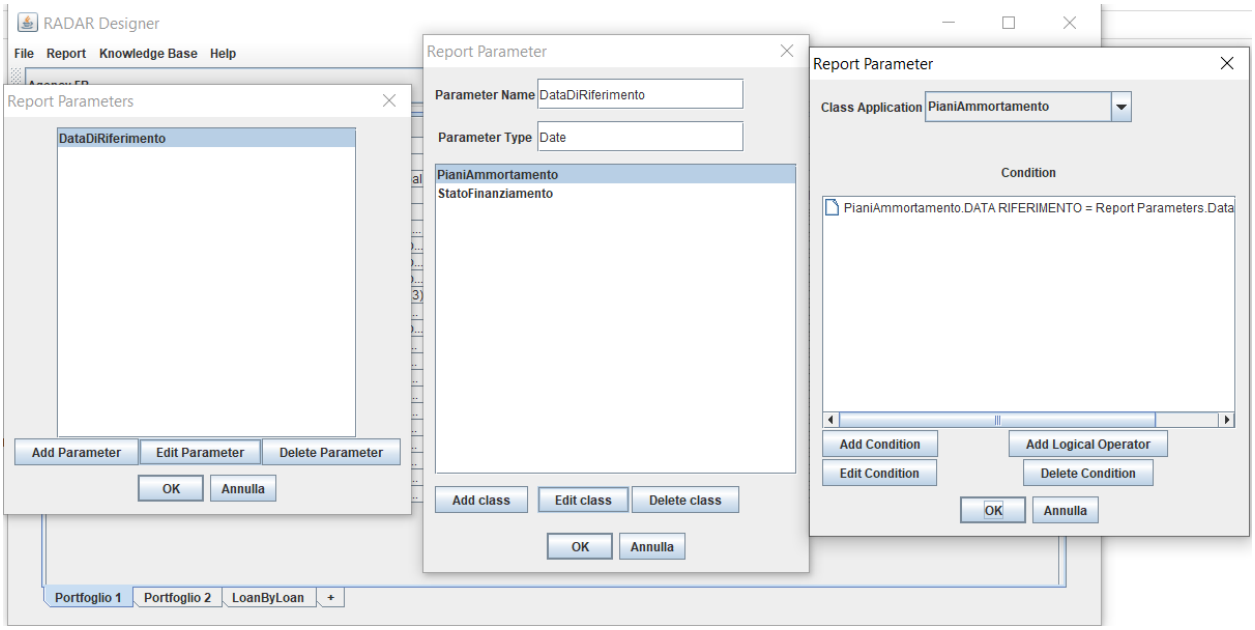

Figure 23: Form di settaggio dei parametri.

in Figura 23, compito del designer è stabilire a quali classi concrete applicare il parametro e la corrispondente regola di calcolo.

Editando il parametro è possibile visualizzarne le informazioni principali quali il nome, il tipo e la classe di riferimento. Editando successivamente la classe è possibile visualizzarne le corrispondenti informazioni riportanti le tabelle a cui fa riferimento e le condizioni applicate associate ad una determinata classe.

Inoltre, attraverso i comandi "Open" e "Save" riportati nel menù principale del framework, è possibile, rispettivamente, effettuare l'apertura e/o il salvataggio del template del report. Il file del report generato avrà l'estensione ".rd". La struttura del file è descritta successivamente in Appendice.

Una sezione del report è composta da una tabella contenuta all'interno del foglio. Una tabella è formata da n x m celle, ed ogni cella è definita da uno dei seguenti tipi:

- Empty Cell
- Label Cell
- Condition Cell
- Aggregation Cell
- Arithmetic Expression Cell

Il tipo "Empty Cell" rappresenta una cella vuota; gli altri tipi sono di seguito dettagliati.

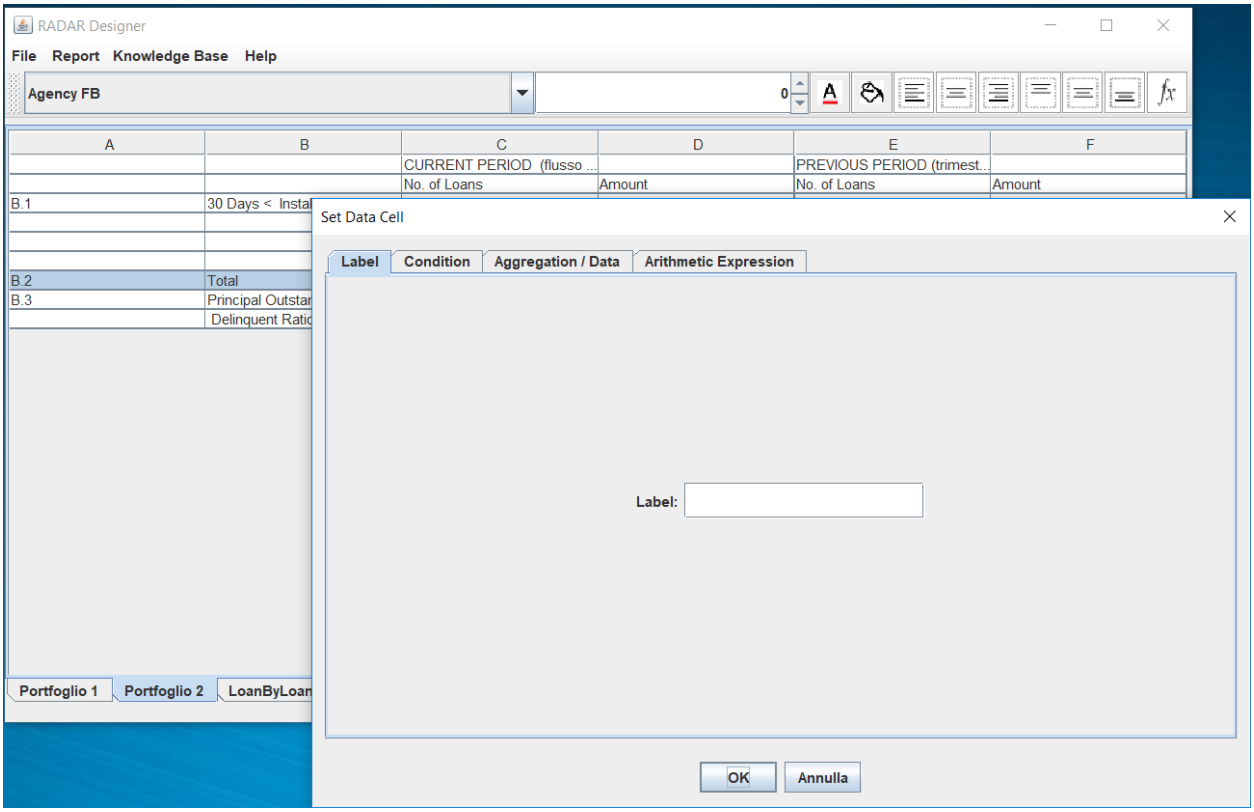

Figure 24: Set cell label

6.1 Set Cell

La preparazione delle interrogazioni/inserimento delle informazioni nelle singole celle del report avviene attraverso una schermata, apribile con un double click sulla singola cella, che possiede i seguenti tab:

- Label
- Condition
- Aggregation
- Arithmetic Expression

6.1.1 Label

La tab **label** permette l'inserimento del testo in una cella. Si veda come riferimento la Figura 24.

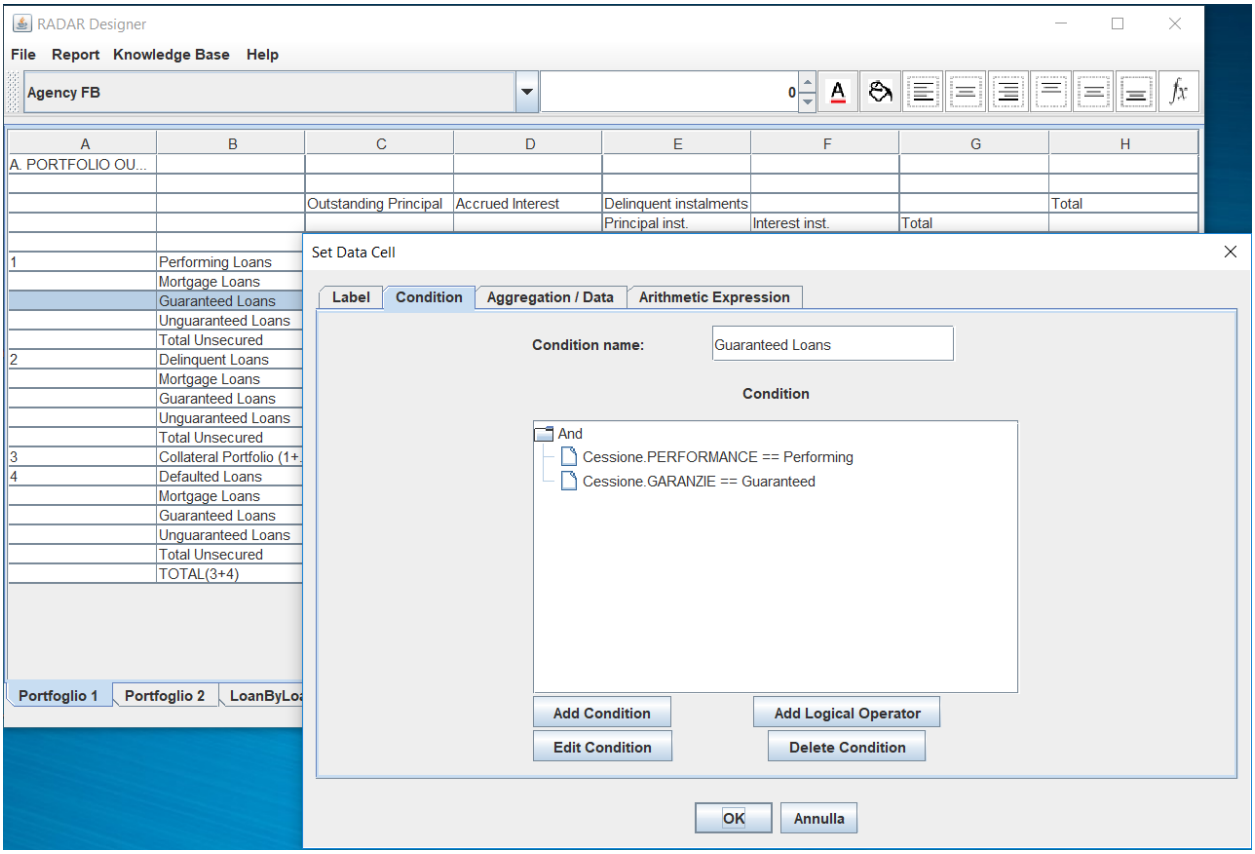

Figure 25: Set cell condition

6.1.2 Condition

La tab **condition** permette di inserire una condizione in una riga del report. La condizione ha validità per tutta la riga della sezione del report, di conseguenza si applicherà la condizione ad ogni aggregazione/espressione aritmetica presente nella stessa riga della condizione.

Semplificazione In una riga può essere presente solo una cella condizione, nel caso ne sia presente più di una il RADAR Engine non genera errore di compilazione, ma i risultati ottenuti in uscita non sono affidabili.

Semplificazione In fase iniziale una condizione è applicata solamente alle righe. In futuro si puè pensare di decidere di applicare una condizione anche ad una colonna della sezione. Nella cella di incrocio riga/colonna sarà ad esempio possibile creare una condizione unita da un "AND logico" ("e").

6.1.3 Aggregation

L'aggregazione serve per eseguire delle aggregazioni (che si traduco in operatori somma, che sommano gli importi delle righe ottenute rispetto ad una proprietà, contatore delle righe (count), calcolo della media (avg), min e max, che ritornano rispettivamente il minimo e il massimo di un gruppo di celle. Un'aggregazione viene inserita attraverso la Form riportata in Figura 26.

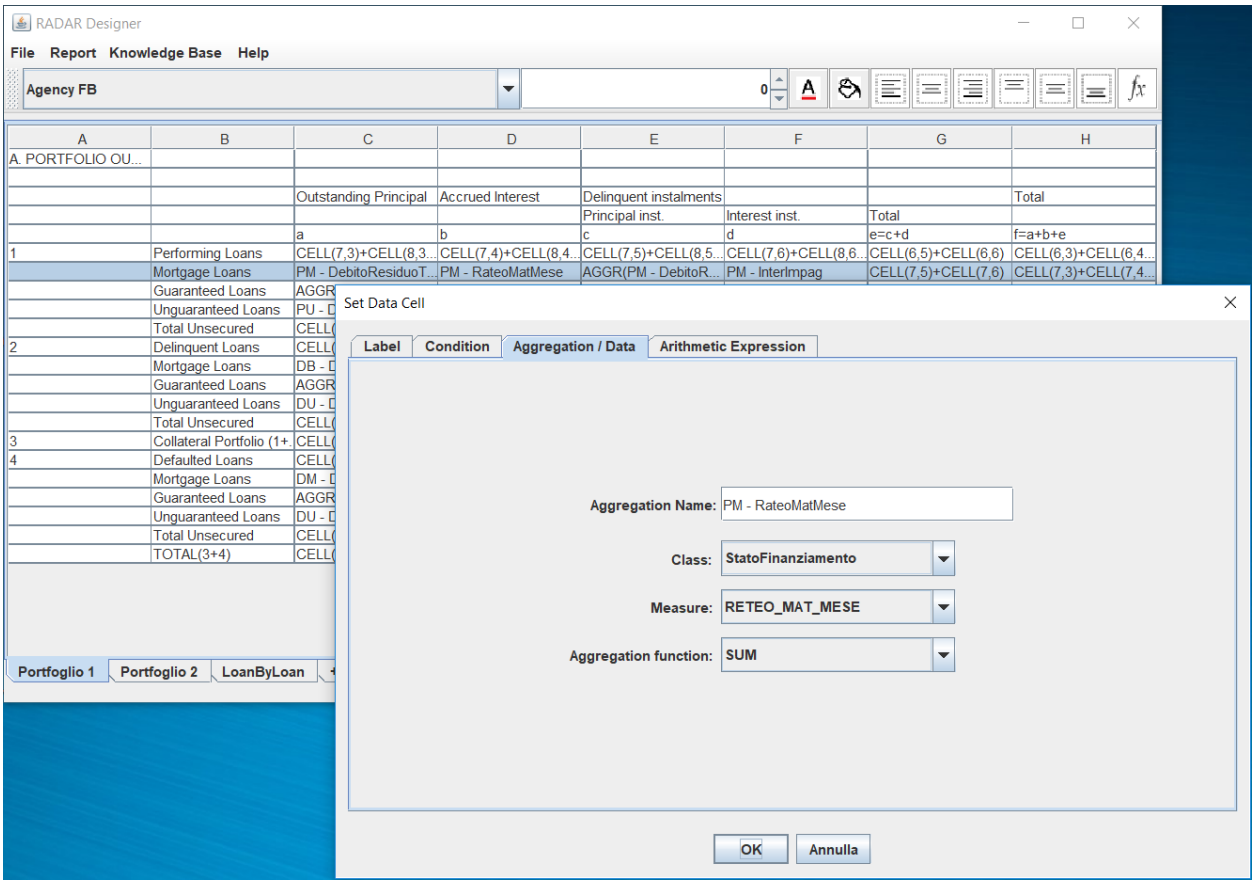

Figure 26: Set Cell Aggregation

6.1.4 Arithmetic Expression

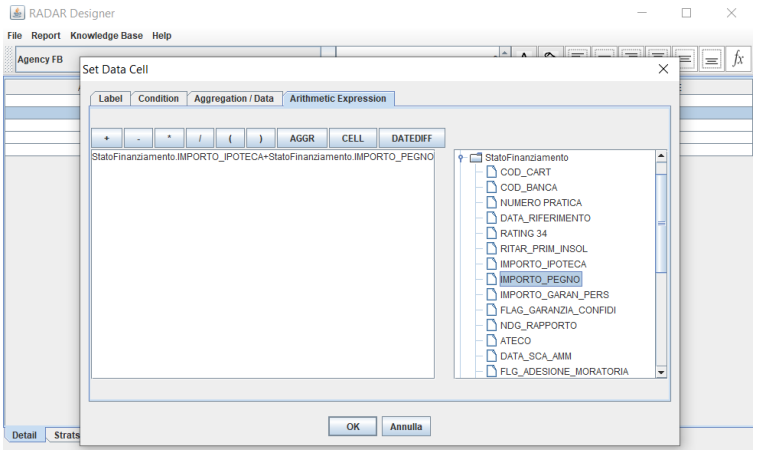

Figure 27: Set cell Arithmetic Expression

La tab Aritmentic Expression mostrata in Figura 27 permette di descrivere espressioni aritmetiche utilizzando valori numerici, aggregazioni o celle precedentemente definite.

Semplificazione: nel progetto realizzato è emersa la necessità di trasformare, esprimendo con terminologia differente, il testo da visualizzare all'interno di una cella: ad esempio può accadere che nel DB siano memorizzati attributi di tipo boolean (true/false), che debbano essere visualizzati in fase di reportistica con la terminologia "SI/NO" oppure "YES/NO". Il problema potrebbe essere risolto inserendo un ulteriore tab all'interno del panel set cell; in alternativa potrebbe essere data maggior espressività al tab aritmetic expression.

Semplificazione Al momento la formattazione del testo avviente rispetto al tipo dell'attributo, può essere neccessario che a tipi uguali corrispondano formattazioni diverse, per esempio ad un double con significato di valuta possano corrispondere al massimo due cifre decimali. Al contrario ad un double utilizzato per esprimere una percentuale si devono considerare anche più di due cifre decimali. Il problema può essere risolto facendo specificare all'utente la formattazione in fase di definizione della tipologia di cella.

6.2 File .rd

Il salvataggio del report creato con l'applicazione RADAR Designer, avviene attraverso un file proprietario strutturato nel seguente modo:

un array di sheets (fogli), dove ogni elemento rappresenta un foglio, Ogni foglio presenta una struttura multipla:

- grid: specifica le dimensioni del foglio excel, gli attributi che contiene l'oggetto sono:
  - rows: intero che indica il numero di righe del foglio
  - columns: intero che indica il numero di colonne del foglio
  - Rows: array di interi che indica l'altezza di ogni riga

  - widthColumns: array di interi che indica la larghezza di ogni colonna

- items:
  - xPosition: numero di riga della tabella del foglio
  - yPosition: numero delle colonne della tabella del foglio
  - type: tipo di cella (Condiction, Aggregation,...)
  - cell: contenuto della cella; esso varia in base al tipo di cella (tipo empty: la cella è vuota, tipo label: la cella contiene la label; tipo condizione: la cella contiene la condizione; tipo aggregazione: la cella contiene i parametri dell'aggregazione; tipo espressione aritmetica: la cella contiene l'espressione aritmetica).
  - style: stile della cella
    * fontType: tipo di font
    * fontSize: dimensione del font
    * fontColor: colore del font
    * backgroundColor: colore dello sfondo della cella
    * TextAlignHorizontal: allineamento orizzontale
    * TextAlignVertical: allineamento verticale

Un estratto del sorgente .JSON è riportato in Appendice.

7 RADAR ETL

In informatica, con i termini estrazione, trasformazione e caricamento (ETL), ci si riferisce alle operazioni necessarie al processo di importazione dati all'interno di un sistema di immagazzinamento dati (database, data warehouse, ...). Nel framework RADAR l'ETL svolge la funzione di trasferimento dati dal db sorgente al target db, nel prototipo il db sorgente è rappresentato da un insieme di file csv.

ETL viene effettuato utilizzando il framework spring batch, rilasciato da Spring. è finalizzato all'implementazione di applicazioni batch per l'elaborazione di grandi volumi di dati, senza alcuna interazione conl'utente. La figura 28 mostra il flusso dello Spring Batch, i cui componenti sono di seguito riportati.

- JobLauncher, componente responsabile dello start dei job
- JobRepository, varifica che il job non sia già stato eseguito e la validità dei parametri, quindi esso è responsabile della conservazione delle informazioni di esecuzione attraverso un repository che può essere in memoria o su DB dedicato.
- Job, è la descrizione del processo che deve essere eseguito, mentre JobInstance è la sua instanza rispetto ai JobParameter, ovvero sui parametri utilizzati dal job. La singola esecuzione del job è rappresentata da una JobExecution: che contiene informazioni sull'esecuzione quali: stato, data ed ora di avvio e conclusione, eventuale eccezione,

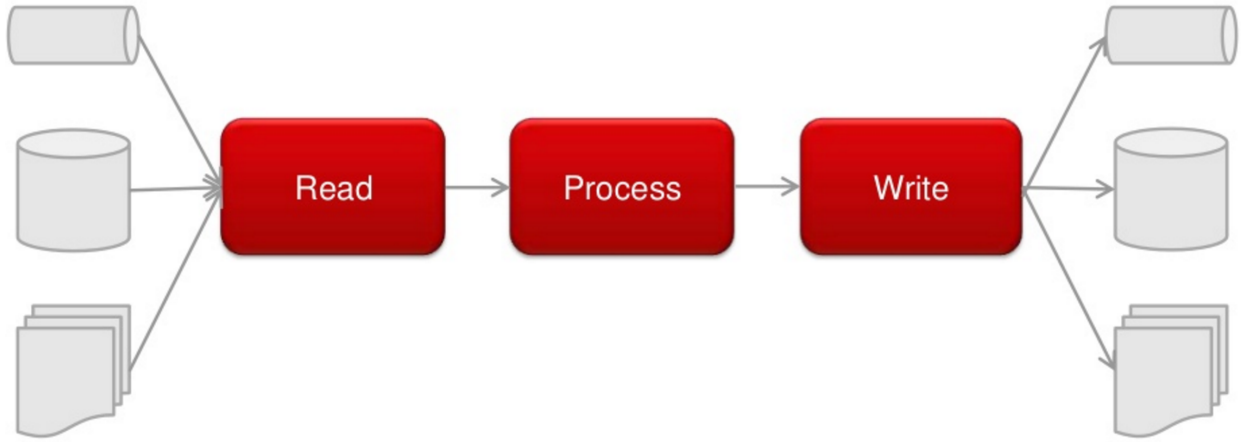

Figure 28: Spring Batch Flow

etc. Poiché una JobInstance può essere eseguita più volte ad essa potranno essere associate più JobExecution. Il Job è suddiviso in una sequenza di Step. Ogni step consiste di tre attività: data reading, processing e data writing. La singola esecuzione di uno step è rappresentata da un oggetto StepExecution che contiene tutte le informazioni di esecuzione: stato, data ed ora di avvio e completamento, numero di letture, scritture e commit, etc. Spring Batch utilizza un meccanismo di elaborazione a chunk (blocchi). Questo implica che i dati vengono letti e processati a blocchi utilizzando l'ItemReader e l'ItemProcessor, quindi vengono aggregati e solamente quando è raggiunto l'intervallo di commit richiesto, l'intero blocco è passato all'ItemWriter e la transazione è committata.

Tra le principali caratteristiche offerte dal framework si evidenzia il supporto a sorgenti di tipo differente, come ad esempio xml, csv, JPA, MongoDb, ecc.

il codice è presente all'interno del packege ETL (org.c2t.ETL). L'ETL viene suddiviso in tre parti:

- Lettore: ha il compito di leggere le informazioni del nostro file, il vantaggio di utilizzare un lettore integrato è che gestisce i guasti delle applicazioni in automatico e supporterà il riavvio in caso di guasto
- Processore: Spring non offre un processore dei dati già integrato ma viene lasciata all'utente l'implementazione. Nel nostro caso non viene applicata nessuna trasformazione dei dati in fase di caricamento.
- Scrittore: salva i dati nel db target

7.1 Configurazione RADAR ETL

Il RADAR ETL è la attività più complicata del RADAR framework, in quanto richiede una precisa configurazione. Partendo dalla informazioni inserite con il KB Designer e contenute nel KB Db, dobbiamo dapprima generare le classi Java di configurazione dell'ETL e il DDL

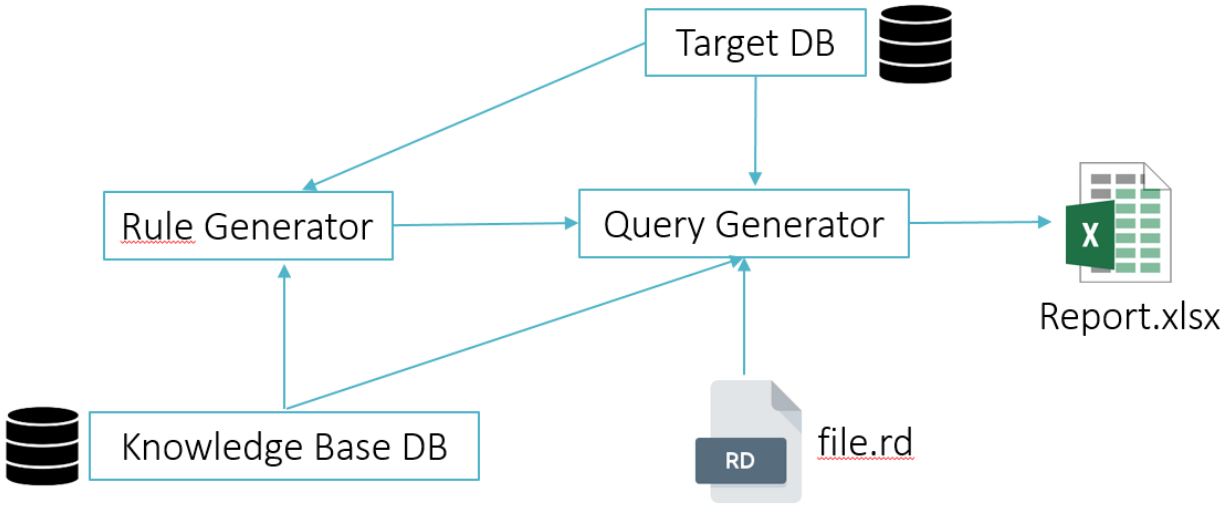

Figure 29: Architettura del Report

per il db target, in un secondo momento effettuare la trasposizione dei dati dal db source al target db.

- DDL Target DB: le istruzioni di DDL per la creazioni del target db avvengo trasformanto il file JSON descritto in Sezione 5.3, contentente la struttura del db. Esso dovrà essere importato manualmente sul server contenente il db, il framework è strutturato in modo che sia indipendente dal particolare dbms di tipo relazionale
- file xml-mapping: Partendo dal file JSON descritto in Sezione 5.3, contentente il mapping tra source db e target db, si generano i file xml di mapping che sono utilizzati dal processore lettore di Spring Batch per l'importazione dei dati dal sourch db.
- bean: Utilizzando sempre il file Json che descrive il db target, si generano i file di bean che svolgo il compito di interfacciare l'ETL al DB. Per avere una generazione automatica dei bean non si utilizza il "formalismo" di scrittura delle variabili secondo la prassi SQL. Per agevolare il compito di generazione del bean, essendo una classe Java utilizziamo Lombok, permettendoci di tralasciare la scrittura dei costruttori e metodi di setter e getter.

8 RADAR Engine

Il RADAR Engine è il cuore del progetto, la cui architettura è riportata in Figura 29. Il suo ruolo è la ricezione in input dei template dei report e la restituzione in output dei report compilati con i risultati delle query/aggregazioni precedentemente scritte. Esso si compone di due attività sequenziali: rules generator e query genertor.

8.1 Rules Generator

Il generatore delle regole, assume la funzione di generare le query sql per svolgere delle operazioni di aggiornamento sul database target dopo l'importazione dei dati da parte dell'ETL. Le regole sono salvate nella rules collection contenuta nel Knowledge Base DB. Ogni regola ha una sua priorità, regole a priorità minore (uno), sono le prime a essere convertite ed eseguite.

Trasformazione di un regola. Una regola può essere convertita in una o più query di update SQL. Ogni azione di una regola corrisponde ad una query (es. regole con tre azioni corrisponderanno tre query). Nel caso di una regola custom, anch'essa presenta una query di update per ogni sua azione. Una regola custom va eseguita a valle della ragola padre a cui appartiene, in quanto prima si aggiornano i dati rispetto alla regola padre, poi si modificano le istanze proprie della regola custom.

Il Freamework RADAR, ha tra i suoi obiettivi di progetto, come precedentemente detto, quello di provare a mantenere la persistenza tra i diversi DBMS, per far ciò è stato impiegato l'ORM (Object Relation Mapping) Hibernate. Esso presenta un proprio dialetto SQL chiamato HQL (Hibernate Query Language), il quale si differenzia principalmente da SQL per una minor espressività del linguaggio. Per esempio non è possibile utilizzare l'istruzione JOIN nelle query di update e delete.

Una semplice query di aggiornamento in HQL presenta la seguente struttura:

```
UPDATE table_name
SET attribute = value
WHERE condition
```

Per ovviare al proplema dell'impossibilità di utilizzo dell'istruzione JOIN, una query di aggiornamento in HQL assume la seguente struttura:

```
UPDATE table_name
SET attribute = value
WHERE condition and (key_table_name_1) IN (
    SELECT key_table_name_1
    FROM table_name_1
    WHERE condition1
    )
```

La condizione di WHERE corrisponde alla condizione salvata nel campo rule della rulesCollection presente nel Knowledge Base DB. Il campo SET corrisponde alla actions. Il nome della tabella è presente nel campo arguments dell'azione.

Durante la generazione della regola bisogna tenere in considerazione i parametri di flusso, cioè quei parametri che esprimono su quali dati presenti del target db è possibile aggiornare i dati.

L'esecuzione del RG, avviene attraverso l'esecuzione di un jar java, esso può ricevere in ingresso, opzionali, come parametri i parametri di contestesto del report,

```
java -jar RADARRules.jar "Codice Cartolarizzazione" 83 "Data di riferimento" "2019-0
```

8.2 Query Generator

Il Query Generator (QG) è l'attività finale del RADAR framework, il suo compito è leggere le informazioni contenute all'interno delle celle (item) del report template (.rd) e convertirle in query HQL. Il QG genera query di selezione e/o di aggregazione, ogni cella contiene una query.

La tipologia delle query varia in base al tipo di report da generare: in un report di stratificazione sono definite query di aggregazione, al contrario in report di dettaglio sono definite query di selezione.

Al fine di ottimizzare i tempi di calcolo delle query ogni report viene analizzato "riga per riga", generando, dove possibile, una query di selezione per ogni riga. In presenza di celle contenenti espressioni aritmetiche, sarà compito di HQL calcolarne il valore.

Esempio di una query di raggruppamento

```
SELECT SUM(attribute)
FROM table_name
WHERE condition
```

L'argomento "attribute" corrisponde alla misura per cui si intende raggruppare. La condizione dell'aggregazione è composta dalla condizione generata dai parametri del report e dalla condizione di riga, nel caso in cui una sua cella sia di tipo condizione.

Semplificazione per migliorare ulteriormente le performances del query generator può essere utile delegare al DBMS il solo calcolo delle query di aggregazione, salvandone in memoria centrale i risultati. In un passo successivo, attraverso una funzione ricorsiva scritta ad hoc, si valorizzeranno le celle contenenti esperessioni aritmetiche.

Al termine di questa fase viene prodotto in uscita il report generato in formato Excel (.xlxs). Di seguito vengono riportati nelle Figure 36,31 e 32 esempi dei report prodotti su differenti fogli Excel, riepilogativi di differenti dettagli in ambito Reportistico Bancario/Finanziario.

L'esecuzione del QG, avviene attraverso l'esecuzione di un jar java, esso riceve come parametro in ingresso il report da compilare e, opzionali, i parametri di contesto del report.

```
java -jar RADARCompileReport.jar "reportEsempio.rd" "DataDiRiferimento" "2019-04-30"
```

9 Compile

10 Problematiche e sviluppi futuri

Il prototipo sviluppato, come precedentemente descritto permette già una generazione di report, anche se è ancora in una fase molto emprionale. Esso mostra ampi margini di miglioramento in quesi tutti i suoi componenti sviluppati. Strutturalmente la fase più critica è la fase di ETL e di strutturazione del database target, in quanto per ogni modifica del db (ALTER TABLE) comporta un rilascio del prototipo. L'impossibilità di non modificare il db target dinamicamente, se da un lato limita i problemi di "consistenza" dei dati, dall'altro

A. PORTFOLIO OUTSTANDING BALANCE							
		Outstanding Princi	Accrued Interest	Delinquent instalments			Total
				Principal inst.	Interest inst.	Total	
		a	b	c	d	e=c+d	f=a+b+e
1	Performing Loans	3.503.625.969,26	9.366.843,66	5.925.218.330,56	184.932,84	5.925.403.263,40	9.438.396.076,32
	Mortgage Loans	1.587.498.994,61	6.261.006,73	3.175.779.552,30	159.756,52	3.175.939.308,82	4.769.699.310,16
	Guaranteed Loans	541.628.330,57	1.222.802,38	223.633,99	17.023,10	240.657,09	543.091.790,04
	Unguaranteed Loans	1.374.498.644,08	1.883.034,55	2.749.215.144,27	8.153,22	2.749.223.297,49	4.125.604.976,12
	Total Unsecured	1.916.126.974,65	3.105.836,93	2.749.438.778,26	25.176,32	2.749.463.954,58	4.668.696.766,16
2	Delinquent Loans	23.980.736,62	218.958,81	684.687,67	167.517,34	852.205,01	25.051.900,44
	Mortgage Loans	22.210.691,18	202.688,80	499.906,66	158.989,29	658.895,95	23.072.275,93
	Guaranteed Loans	1.265.029,17	15.015,94	94.967,86	5.230,86	100.198,72	1.380.243,83
	Unguaranteed Loans	505.016,27	1.254,07	89.813,15	3.297,19	93.110,34	599.380,68
	Total Unsecured	1.770.045,44	16.270,01	184.781,01	8.528,05	193.309,06	1.979.624,51
3	Collateral Portfolio (1+2	3.527.606.705,88	9.585.802,47	5.925.903.018,23	352.450,18	5.926.255.468,41	9.463.447.976,76
4	Defaulted Loans	36.666.792,29	412.802,67	80.502.182,12	425.076,20	80.927.258,32	118.006.853,28
	Mortgage Loans	26.259.794,21	283.560,04	63.904.256,71	285.072,41	64.189.329,12	90.732.683,37
	Guaranteed Loans	3.076.754,25	35.887,36	1.937.437,75	55.113,67	1.992.551,42	5.105.193,03
	Unguaranteed Loans	7.330.243,83	93.355,27	14.660.487,66	84.890,12	14.745.377,78	22.168.976,88
	Total Unsecured	10.406.998,08	129.242,63	16.597.925,41	140.003,79	16.737.929,20	27.274.169,91
	TOTAL(3+4)	3.564.273.498,17	9.998.605,14	6.006.405.200,35	777.526,38	6.007.182.726,73	9.581.454.830,04

Figure 30: Report di output "Portfolio 1".

		CURRENT PERIOD (flusso del trimestre)		PREVIOUS PERIOD (trimestre precedente)			
		No. of Loans	Amount	No. of Loans	Amount		
B.1	30 Days < Installment <= 59 Days	101,00	13.450.186,22				
	60 Days < Installment <= 89 Days	58,00	6.312.256,92				
	90 Days < Installment <= 119 Days	04,00	3.602.224,19				
	120 Days < Installment <= 179 Days	08,00	1.687.233,11				
B.2	Total	171,00	25.051.900,44				
B.3	Principal Outstanding Balance of Loans at closing date						
	Delinquent Ratio	CELL(7,3)/CELL(8,3	CELL(7,4)/CELL(8,4)				

Figure 31: Report di output "Portfolio 2".

Portfolio Loan-by-Loan

NDG rapporto	Loan ID	Outstanding Balance	Current Rating	Loan Status*	Risk Segmentation	Internal Probab	Mortage Amount	Indexed Property	Renegotiations (#	MORATORIA	FINANZIAMEN	Time to Maturit	WAL (in years)	Cod. ATECO
7167361	1002923	00,00	4	3	2630	00,00	00,00	00,00	0			-111		256200
7018569	1002928	2.776.094,43	4	3	2630	00,00	00,00	00,00	0			1007		432201
7140360	1002970	00,00	2	3	2520	00,00	00,00	00,00	0			-157		681000
4572884	1002975	100.661,82	4	3	2630	00,00	00,00	00,00	0			1002		331230
12009907	1002996	7.367,44	6	3	2500	00,00	00,00	00,00	0			957		563000
8471278	1003044	503.368,83	3	3	2630	00,00	00,00	00,00	0			272		451101
8864503	1003130	00,00	3	3	2530	00,00	00,00	00,00	0			-156		467600
8435762	1003148	29.660,55	4	3	2530	00,00	00,00	00,00	0			213		475210
7304874	1003170	1.875,59	6	3	2500	00,00	00,00	00,00	0			221		812000
8471552	1003172	221.694,63	5	3	2630	00,00	00,00	00,00	0			630		234200
7298619	1003183	16.320,92	4	3	2500	00,00	00,00	00,00	0			578		464500
7288087	1003215	00,00	3	3	2630	00,00	00,00	00,00	0			-152		352300
7357034	1003245	432.882,79	6	3	2520	00,00	1.000.000,00	1.186.603,00	0			4597		681000
4692209	1003294	357.849,26	5	3	2630	00,00	00,00	00,00	0			684		631119
7279771	1003301	00,00		3	2500	00,00	00,00	00,00	0			-336		464500
8868512	1003315	28.929,62	6	3	2530	00,00	00,00	00,00	0			938		477700
8873976	1003432	1.454.681,84	5	3	2530	00,00	3.200.000,00	3.917.621,00	0			3922		823000
8429844	1003470	8.968,19	2	3	2540	00,00	00,00	00,00	0			944		439909
7117286	1003477	109.763,22	6	3	2530	00,00	00,00	00,00	0			946		433100
8452887	1003494	28.167,95	5	3	2630	00,00	00,00	00,00	0			279		494100

Figure 32: Report di output "LoanByLoan".

non permette quella dinamicità utile per adattarsi velocemente ai diversi contesti applicativi. La soluzione attualmente adottata prevede l'utilizzo di un database relazione, ma il tutto viene reso "statico", si potrebbe pensare di adottare invece una soluzione NoSQL per dare quella dinamicità che manca.

Una soluzione possibile, potrebbe essere di apportare una modifica alla configurazione di Hibernate. Infatti attraverso la libreria ClassLoader si potrebbe pensare di passare dinamicamente ad Hibernate le entità che esso successivamente mappera sul database target. Hibernate permete di apportare sui database solo modifiche di update, create, ma mai di delete. Di conseguenza l'eliminazione di una proprietà dalla Knowledge Base, non attuerà nessuna modifica al database, questo problema non è così limitato in quanto una operazione di delete è sempre una operazione complessa e potenzialmente pericolosa.

11 Conclusioni

Come descritto nell'introduzione, le attività di reportistica richieste agli istituti finanziari sono necessarie per differenti scopi, ad eseempio per uso interno, per la revisione contabile e per i processi decisionali. Di fatto, le autorità istituzionali e di regolamentazione, come le banche centrali e le agenzie di rating, impongono agli istituti finanziari di fornire periodicamente report che illustrino l'attuale situazione finanziaria della loro attività, in modo da verificarne la conformità rispetto alla normativa vigente. I collaboratori coinvolti in queste attività devono quindi interpretare sistematicamente i requisiti interni ed esterni sulla base dei dati disponibili. In tl senso, uno dei principali punti critici delle organizzazioni riguarda la mancanza di dati integrati e la mancanza di un dizionario di dati standard.

Il framework implementato, RADAR (acronimo di Rich Advanced Design Approach for Reporting) parte dalla nozione di Operational Data Store (ODS) [7, 15], cioè un database che fornisce una visione unificata dei dati, e si pone come parte integrante fra un'ontologia di termini e concetti e l'effettivo schema operativo (e relazionale) dei dati di origine. Una volta definito, lo schema relazionale "RADAR Schema" (così chiamato perché basato sul modello dei dati "RADAR Data Model") fornisce una visione di alto livello dei dati di origine sulla base dei concetti descritti nell'ontologia per lo specifico contesto applicativo. In altre parole, esso fornisce una visione concreta dei concetti sulla base dei dati effettivi memorizzati all'interno dell'ODS su cui lavora il framework per i dipendenti. Ciò permette loro di avere una visione dei dati semanticamente migliore: essi infatti possono utilizzare questo Schema per specificare le misure aggregate da inserire nei report, sfruttando i termini provenienti dall'ontologia, e definire in tal modo al meglio i dati da utilizzare per la generazione della documentazione finale (report).

Come precedentemente riportato, anche se l'approccio proposto è stato applicato al contesto finanziaro, esso è stato definito come un framework generale e, di conseguenza, può essere considerato ed applicato anche in contesti differenti da quello descritto nel presente progetto.

12 Appendice

12.1 Classes Collection

Contiene tutte le classi concrete da noi definite


```
{
    "_id":"5ca4b8b90e474767f43cfedd",
    "interpretationName":"PianiAmmortamento",
    "subClassOf": "[Thing, Intangible, Service, FinancialProduct, LoanOrCredit, Pian
    "properties":[
        {
            "Property":"CODICE ABI",
            "Expected Type":"Integer",
            "Description":"",
            "Attribute type":"Categorico",
            "Key":true,
            "ofClass":"PianiAmmortamento"
        },
        {
            "Property":"PERFORMANCE",
            "Expected Type":"Text",
            "Description":"",
            "Attribute type":"Categorico finito",
            "Key":false,
            "ofClass":"StatoFinanziamento",
            "valueAttributeType":[
                {
                    "name":"Performing"
                },
                {
                    "name":"Delinquent"
                },
                {
                    "name":"Defaulted"
                }
            ]
        },
        {
            "Property":"QUOTA INTERESSI",
            "Expected Type":"Number",
            "Description":"",
            "Attribute type":"Misura",
            "Key":false,
            "ofClass":"PianiAmmortamento"
```

```
        }
    ]
}
```

12.2 Relations Classes Collection

```
{
    "_id":"5cb5aa9f0e47475d20fd8772",
    "nameRelation":"Relazione_Cessione",
    "classFrom":"StatoFinanziamento",
    "classTo":"Cessione",
    "relationProperties":[
        {
            "propertyFrom":"COD_CART",
            "propertyTo":"CODICE CARTOLARIZZAZIONE"
        },
        {
            "propertyFrom":"COD_BANCA",
            "propertyTo":"CODICE ABI"
        },
        {
            "propertyFrom":"NUMERO PRATICA",
            "propertyTo":"NUMERO PRATICA"
        }
    ]
}
```

12.3 Rules Collection

```
{
    "_id":"5cb9cfda0e4747237ce81952",
    "ruleName":"NonEstinto",
    "rule": {
        "userObject": {
            "argumentOne": "StatoFinanziamento.DEBITO_RESIDUO_EFFETTIVO",
            "comparator": "=",
            "argomentTwo": "0"
        },
        "children": []
    },
    "priority":1,
    "actions":[
        {
            "argument":"(Cessione via Relazione_Cessione).Estinto",
```

```
            "tag":"0"
        }
    ],
    "customizedRule":[]
}
```


12.4 Context Parameter Collection


```
{
    "_id":"5d02164ce1f27b1710caf00f",
    "name":"Codice Banca",
    "properties":[
        {
            "namePropertie":"PianiAmmortamento.CODICE ABI"
        },
        {
            "namePropertie":"Cessione.CODICE ABI"
        },
        {
            "namePropertie":"StatoFinanziamento.COD_BANCA"
        }
    ]
}
```


12.5 Mapping Collection


```
{
    "_id":"5cb5af170e47475c84a21e04",
    "interpretation":"PianiAmmortamento",
    "mappingTable":"pianiAmmortamento",
    "mapping":[
        {
            "property":"CODICE ABI",
            "attribute":"CODICE_ABI"
        },
        {"property":"CODICE CARTOLARIZZAZIONE",
        "attribute":"CODICE_CARTOLARIZZAZIONE"
        },
        {
            "property":"DATA RIFERIMENTO",
            "attribute":"DATE_RIFERIMENTO"
        },
        {
            "property":"RAPPORTO",
            "attribute":"RAPPORTO"
```

```
            },
            {
                "property":"PROGRESSIVO",
                "attribute":"PROGRESSIVO"
            },
            {
                "property":"DATA SCADENZA",
                "attribute":"DATE_SCADENZA"
            },
            {
                "property":"DATA ESIGIBILITA",
                "attribute":"DATE_ESIGIBILITA"
            },
            {
                "property":"QUOTA CAPITALE",
                "attribute":"QUOTA_CAPITALE"},
            {
                "property":"QUOTA INTERESSI",
                "attribute":"QUOTA_INTERESSI"
            }
        ]
    }
```

12.6 Collection del Rule Designer

Dizionario che contiene tutte le regole, specificando per ogni proprietà dell'interpretazione le regole di calcolo

```
{
    "ruleName": "Performing",
    "rule": {
        "userObject":{
            "argomentOne": "StatoFinanziamento.RATING 34"
            "comparator": "==",
            "argomentTwo": "10"
        },
        "children":[]
    },
    "priority": 1,
    "actions": [
        {
            "argument": "(Cessione via Relazione_Cessione).PERFORMANCE"
            "tag": "Performing"
        }
    ]
```

```
}
```

12.7 Frammento file .rd

CODICE RIPRESO DALLA SEZIONE FILE.RD DEL REPORT TEMPLATE

```
{
  "sheets": [
    {
        //sheet one
        "grid":
            {
                "rows": 10,
                "columns"s: 10,
                "widthRows": 42,
                "heightColumns": 18
            },
        "items": {
            [
                {
                    //item
                    "position": [1,1],
                    "properties":{
                        "type": "header",
                        "text": "Performing Loans"
                    },
                    "style":
                        {
                            "fontType": "Arial",
                            "fontColor": #000000,
                            "fontSize": 12,
                            "backgroundColor": #ffffff,
                            "textAlignHorizontal": "left",
                            "textAlignVertical": "center"
                        }
                },
                {
                    //item
                    "position": [1,2],
                    "properties":{
                        "type": "aggregazione",
                        "etichettaAggregazione": "Performing Loans",
                        "misura": 784388232.49,
```

```
        "descrizione": "debito residuo"
    },
    "style":
        {
            "fontType": "Arial",
            "fontColor": #000000,
            "fontSize": 12,
            "backgroundColor": #ffffff,
            "textAlignHorizontal": "left",
            "textAlignVertical": "center"
    }
},
{
    //item
    "position": [1,3],
    "properties":{
        "type": "dettaglio",
    },
    "style":
        {
            "fontType": "Arial",
            "fontColor": #000000,
            "fontSize": 12,
            "backgroundColor": #ffffff,
            "textAlignHorizontal": "left",
            "textAlignVertical": "center"
},
{
    "position": [1,4],
    "properties":{
        "type": "blank",
    },
    "style":
        {
            "fontType": "Arial",
            "fontColor": #000000,
            "fontSize": 12,
            "backgroundColor": #ffffff,
            "textAlignHorizontal": "left",
            "textAlignVertical": "center"
},
{
    //item
    ...
}
```

```
            ]
        }
    },
    {
        //sheet two
        ...
    }
  ]
}
```

13 Appendice B: Esecuzione dell'Applicazione

L'esecuzione dell'applicazione richiede l'installazione di Mongo DB e della Java Virtual Machine (JVM) per l'esecuzione di eseguibili con estensione .jar.

NICOLA: NON E' VERO. FACILITA SOLO L'UTILIZZO

L'esecuzione dell'applicazione allo stato attuale prevede che il Database di partenza del Knowledge Base sia popolato.

Per l'esecuzione dell'attività di Mapping, come anche precedentemente riportato, è necessario caricare due sorgenti: .csv e .json.

Attualmente la generazione del report finale avviene eseguendo il main RADAR Engine di query e report generator a parte utilizzando in ingresso il file con estensione .rd.

Il file con estensione ".rd" viene creato dal Knowledge Designer (attraverso la fase di salvataggio delle uscite prodotte). Mentre il KB Designer è gestito da un' interfaccia grafica, come riportato in Figura 3, le cui form sono state sopra descritte attraverso vari screenshots dettagliati, i moduli di ETL, Query e Report generator, così come il configuratore di ETL non hanno alcuna interfaccia grafica. Il flusso di questi moduli è rappresentato principalmente nell'Information System Layer della Figura 3, così come gli ingressi e le uscite prodotti da ciascun modulo.

Il report finale viene infine generato dal RADAR Engine, responsabile, come sopra descritto, della generazione del report attraverso l'esecuzione del modulo "Report Generator".

14 Appendice C: Installation Guide

Il primo passo da compiere è l'installazione dei database per l'archivio delle informazioni. I DBMS possono risiedere sulla stessa macchina di esecuzione del sistema o in una macchina dedicata. I database neccessari sono:

- MongoDb: per la Knowledge Base DB. Per l'installazione si rimanda alla guida ufficiale: https://docs.mongodb.com/v3.2/tutorial/
- DBMS per l'archiviazione dei dati da analizzare. La scelta del particolare DBMS è lasciata al committente. Esso deve ricadere tra quelli supportati da Hibernate, tra i

```java
public class MongoDbMDConfiguration {

    public static String mongoDbURL = "mongodb://localhost:27017";
    public static String mongoDbMDdb = "MDdb";
    public static String mongoDbClassesCollections = "ClassesCollection";
    public static String mongoDbMappingCollections = "MappingCollection";
    public static String mongoDbRulesCollections = "RulesCollection";
    public static String mongoDbRelationClassCollections = "RelationsClassesCollection";
    public static String mongoDbContexCollections = "ContexCollection";
    public static String mongoDbContexParameterCollections = "ContexParameterCollection";
}
```

Figure 33: Esempio di configurazione MongoDbMDConfiguration

```xml
    <bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
        <property name="driverClassName" value="com.mysql.jdbc.Driver" />
        <property name="url" value="jdbc:mysql://localhost:3306/RADAR" />
        <property name="username" value="root" />
        <property name="password" value="" />
    </bean>
```

Figure 34: Esempio di configurazione jobx.xml.

principali citiamo: Oracle Db, DB2, Microsoft SQL Server, Sybase, MySQL e PostgreSQL.

Per terminare la preparazione dell'ambiente di esecuzione è neccessaria l'installazione della Java Virtual Machine (JVM), scaricabile da: https://www.java.com/it/download.

Configurazione dei database

I dati di configurazione dei database sono da inserire nei seguenti file:

- MongoDbMDConfiguration: classe situata nel package org.c2t.mongoDB, inidicare nella variabile mongoDbURL l'indirizzo di collocazione del database Knoledge Base DB. Figura 33
- jobx.xml, file di configurazione di Spring Batch. Indicare nel bean dataSourch i dati di connessione al targetDb (indirizzo, nome utente e password) e il tipo di DBMS. Figura 34
- hibernate.cfg.xml file di configurazione di Hibernate. Indicare nel bean dataSourch i dati di connessione al targetDb (indirizzo, nome utente e password) e il tipo di DBMS. Figura 35

L'utilizzo del RADAR Designer avviene attraverso l'esecuzione del jar RADARDesigner.jar, avviabile con il seguente comando da command line:

```
java -jar RADARDesigner.jar
```

```
<hibernate-configuration>
    <session-factory>
        <!-- Database connection settings -->
        <property name="connection.driver_class">com.mysql.jdbc.Driver</property>
        <property name="connection.url">jdbc:mysql://localhost:3306/RADAR</property>
        <property name="connection.username">root</property>
        <property name="connection.password"></property>

        <property name="hibernate.dialect">org.hibernate.dialect.MySQLInnoDBDialect</property>
        <property name="hibernate.show_sql">false</property>
        <property name="hibernate.connection.autocommit">false</property>
        <property name="hibernate.format_sql">true</property>
        <property name="hibernate.temp.use_jdbc_metadata_defaults">false</property>

        <mapping class="org.c2t.ETL.bean.pianiAmmortamentoBean" />
        <mapping class="org.c2t.ETL.bean.statoFinanziamentoBean" />
        <mapping class="org.c2t.ETL.bean.cessioneBean" />

    </session-factory>
</hibernate-configuration>
```

Figure 35: Esempio di configurazione hibernate.cfg.xml.

Alla prima installazione il KB Designer, al momento incluso nel RADAR Designer, presenta solo l'ontologia di base Schema.org, Al contrario, se vogliamo rilasciare una versione arricchita nel dizionario va importata all'interno del KB DB in MongoDb secondo lo schema precedentemente descritto.

Configurazione ETL

Dopo che l'utente ha definito le classi, relezioni e il mapping, può essere avviato il processo di configurazione dell'ETL, e del RADAR Engine. Per facilitare la fase di configurazione è stato realizzato un script RadarConfiguration.jar

In input riceve:

- le informazioni di mapping contenute nel Konwledge Base Db
- la struttura del data source, contenente le informazioni dei tipi dei campi
- la posizione dei campi da leggere dalla sorgente dati

Esso produrra in output una serie di file di configurazione:

- XML-Mapping: un file per ogni classe concreta descritta dall'ontologia, contiene la mappatura con la sorgente dati
- Java Bean: una classe per ogni classe concreta, contiene la mappatura con il target db
- DDL: file da eseguire nel DBMS, contiene la struttura del target db

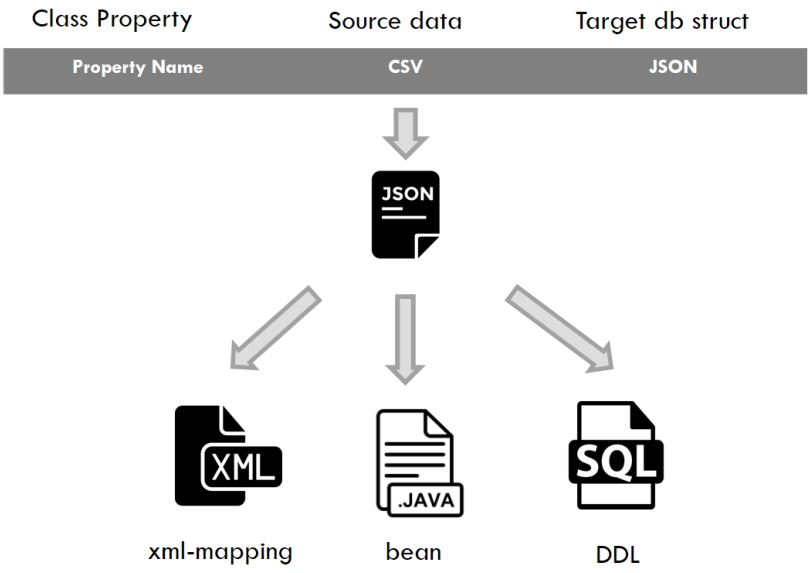

Figure 36: Report di output "Portfolio 1".

Esecuzione ETL

La fase di importazione dei dati avviene con l'esecuzione di RADARETL.jar, non necessita di nessun parametro di input, in quanto la sorgente dati viene mappata durante la configurazione dell'ETL

```
java -jar RADARETL.jar
```

Esecuzione delle Regole

Ogni volta che si importano nuovi dati all'interno del target db o che vengono create / modificate delle regole è neccessario l'aggiornamento dei dati rispetto alle regole dichiarate. L'aggiornamento avviene attraverso l'esecuzione del RADARRules.jar esso può ricevere in input opzionali dei parametri di contesto, cioè dei parametri per limitare l'esecuzione ad un set ristretto di dati presenti nel target db. La coppia [Parametro, valore del parametro] è array.

```
java -jar RADARRules.jar "Codice Cartolarizzazione" 83 "Data di riferimento" "2019-0
```

Compilazione del report

La compilazione del report avviene attraverso l'esecuzione del jar RADARCompileReport.jar esso necessita della ricezione di uno o più parametri passati in fase di avvio nel seguente ordine:

- Indirizzo del template del report (.rd)
- $[Parametrodelreport, valoredelparametro]*$

  opzionale, array contenente una coppia nome - valore di parametri del report

L'esecuzione avviene eseguento il jar da linea di comando nel seguente modo:

```
java -jar RADARCompileReport.jar "reportEsempio.rd" "DataDiRiferimento" "2019-04-30"
```

14.1 Pre configurazione della base di conoscenza

All'avvio dell'applicativo dopo l'installazione il dizionario conterrà solo i concetti (classi) dell'ontologia di base caricata, in questo caso Schema.org. I concetti personalizzati, cioè quelli scritti da noi, vanno importati in una seconda fase nella Knowledge Base DB (MongoDB), insieme alle regole e al mapping. Il comando da utilizzare è mongoimport, al seguente link la documentazione specifica del comando:

https://docs.mongodb.com/manual/reference/program/mongoimport/

Un esempio di uso del comando per il caricamente delle classi nella colletion classes è il seguente:

```
mongoimport --db MdDB --collection ClassesCollection --file classes.json
```

Allo stesso modo è possibile importare delle preconfigurazioni per tutte le altre collection:

- MappingCollection
- RulesCollection
- RelationsClassesCollection
- ContexCollection
- ContexParameterCollection

15 Appendice D: User Guide

15.1 Knowledge Base Designer

15.2 Report Designer

Creazione nuovo report

La creazione del nuovo report avviene attraverso l'apertura del menu "impostazioni del report", in Report/new Report.

Comandi per eseguire:
browsing di schema.org
Avvio del radar framework
generazione del report finale

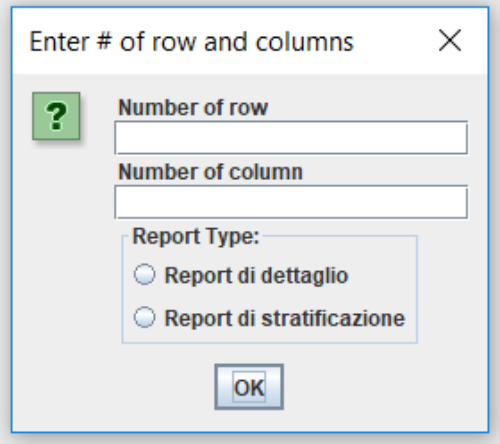

Figure 37: Impostazioni del report

References

[1] Abramowicz, W., Auer, S., Heath, T.: Linked data in business. Business & Information Systems Engineering **58**(5), 323–326 (Oct 2016). https://doi.org/10.1007/s12599-016-0446-0, `https://doi.org/10.1007/s12599-016-0446-0`

[2] The MongoDB General Manual. `https://docs.mongodb.com/manual/`

[3] Berners-Lee, T., Hendler, J., Lassila, O., et al.: The semantic web. Scientific american **284**(5), 28–37 (2001)

[4] Browne, O., O'Reilly, P., Hutchinson, M., Krdzavac, N.: Distributed data and ontologies: An integrated semantic web architecture enabling more efficient data management. Journal of the Association for Information Science and Technology **1** (2019)

[5] Butler, T., Abi-Lahoud, E.: A mechanism-based explanation of the institutionalization of semantic technologies in the financial industry. In: Bergvall-Kåreborn, B., Nielsen, P.A. (eds.) Creating Value for All Through IT. pp. 277–294. Springer Berlin Heidelberg, Berlin, Heidelberg (2014)

[6] Consortium, W.W.W., et al.: Json-ld 1.0: a json-based serialization for linked data (2014)

[7] Golfarelli, M., Rizzi, S., Cella, I.: Beyond data warehousing: what's next in business intelligence? In: Proceedings of the 7th ACM international workshop on Data warehousing and OLAP. pp. 1–6. ACM (2004)

[8] Gruber, T.R.: A translation approach to portable ontology specifications. Knowledge acquisition **5**(2), 199–220 (1993)

[9] Guerrini, M., Possemato, T.: Linked data: un nuovo alfabeto del web semantico. Biblioteche oggi: Mensile di informazione aggiornamento dibattito **30**(3), 7–15 (2012)

[10] Guha, R.V., Brickley, D., Macbeth, S.: Schema. org: evolution of structured data on the web. Communications of the ACM **59**(2), 44–51 (2016)

[11] Kärle, E., Fensel, A., Toma, I., Fensel, D.: Why are there more hotels in tyrol than in austria? analyzing schema. org usage in the hotel domain. In: Information and Communication Technologies in Tourism 2016, pp. 99–112. Springer (2016)

[12] Neches, R., Fikes, R.E., Finin, T., Gruber, T., Patil, R., Senator, T., Swartout, W.R.: Enabling technology for knowledge sharing. AI magazine **12**(3), 36 (1991)

[13] Palma, G., Vidal, M.E., Raschid, L., Thor, A.: Exploiting semantics from ontologies and shared annotations to find patterns in annotated linked open data. In: Proceedings of the 3rd International Conference on Linked Science - Volume 1116. pp. 15–29. LISC'13, CEUR-WS.org (2013)

[14] Petrova, G., Tuzovsky, A., Aksenova, N.: Application of the financial industry business ontology (fibo) for development of a financial organization. In: Proceedings of the International Conference on Information Technologies in Business and Industry - Conference Series of Journal of Physics. IOP Publishing (2016)

[15] Sen, A., Sinha, A.P.: A comparison of data warehousing methodologies. Communications of the ACM **48**(3), 79–84 (2005)

[16] Şimşek, U., Kärle, E., Holzknecht, O., Fensel, D.: Domain specific semantic validation of schema. org annotations. In: International Andrei Ershov Memorial Conference on Perspectives of System Informatics. pp. 417–429. Springer (2017)

[17] Şimşek, U., Kärle, E., Holzknecht, O., Fensel, D.: Domain specific semantic validation of schema.org annotations. In: Petrenko, A.K., Voronkov, A. (eds.) Perspectives of System Informatics. pp. 417–429. Springer International Publishing (2018)

[18] Sporny, M., Longley, D., Kellogg, G., Lanthaler, M., Lindström, N.: Json-ld 1.0. W3C Recommendation **16** (2014)

[19] Swartout, B., Patil, R., Knight, K., Russ, T.: Toward distributed use of large-scale ontology. In: Proceedings of the Tenth Workshop on Knowledge Acquisition for Knowledge-Based Systems. pp. 138–148 (1996)

[20] Zaniolo, C.: A unified semantics for active and deductive databases. In: Rules in database systems, pp. 271–287. Springer (1994)

[21] Şimşek, Umutcan and Kärle, Elias and Holzknecht, Omar and Fensel, Dieter: Domain specific semantic validation of schema.org annotations, International Andrei Ershov Memorial Conference on Perspectives of System Informatics, pp. 417–429, 2017, Springer.