

Università degli Studi di Salerno
Dipartimento di Informatica



Corso di Laurea in Informatica

TiveJS

Un'applicazione Javascript per il riconoscimento di
linguaggi diagrammatici

A Javascript application for the recognition of
Diagrammatic languages

Relatori

Prof. Gennaro Costagliola
Dott. Mattia De Rosa

Candidato

Dario Tecchia
Matr. 0512102581

Questa tesi la dedico alla mia famiglia.

Ringraziamenti

Abstract

La comunicazione visiva è in molti casi più diretta ed immediata rispetto alla comunicazione verbale: disegni, foto e mappe sono esempi di frasi visive che necessitano di un contesto per essere descritte in modo naturale.

In questa tesi presento TiveJS, un'estensione della piattaforma draw.io, che sfrutta simboli e definizioni sematiche per il riconoscimento dei linguaggi diagrammatici e la traduzione di questi in altri linguaggi. Il tool applica delle definizioni semantiche ad un diagramma e restituisce una traduzione di quest'ultimo. La traduzione avviene attraverso due fasi principali: il riconoscimento del grafo e l'applicazione delle definizioni.

Il mio lavoro di tesi si basa su strumenti precedentemente sviluppati: LoCoModeler e Tive. Precedentemente suddiviso in lato client e lato server, Tive è stato re-implementato completamente in JavaScript, prendendo il nome di TiveJS, eliminando così la necessità del server.

Indice

| | |
|-----------------------------------------|------------|
| Ringraziamenti | ii |
| Abstract | iii |
| Elenco delle figure | vi |
| Elenco delle tabelle | vii |
| 1 Introduzione | 1 |
| 1.1 Motivazioni | 1 |
| 1.2 Organizzazione della Tesi | 2 |
| 2 Lavori Correlati | 3 |
| 2.1 Draw.io e mxGraph | 3 |
| 2.2 LoCoMoTiVE | 4 |
| 2.2.1 LoCoModeler | 4 |
| 2.2.2 TiVE | 5 |
| 2.3 DrawSE | 6 |
| 3 Linguaggi visuali | 8 |
| 3.1 Linguaggio verbale | 8 |
| 3.2 Componenti | 8 |
| 3.3 Vantaggi | 9 |
| 4 Local Context | 10 |
| 4.1 Sintassi | 10 |
| 4.2 Semantica | 12 |
| 5 TiveJS | 15 |
| 5.1 Funzionamento | 15 |

| | |
|-----------------------------------------------|-----------|
| Indice | v |
| 5.1.1 Progettazione sentenze visive | 15 |
| 5.1.2 Riconoscimento grafo | 15 |
| 5.1.3 Applicazione definizioni | 15 |
| 5.1.4 Traduzione | 15 |
| 5.2 Implementazione | 15 |
| 5.2.1 Tecnologie usate | 15 |
| 6 Contesti d'utilizzo | 16 |
| 6.1 Entity Relationship | 16 |
| 6.2 Flowchart | 16 |
| 6.3 Tree | 16 |
| 7 Conclusioni | 17 |
| 7.1 Sviluppi futuri | 17 |
| Appendice A Codici | 18 |
| Bibliografia | 19 |

Elenco delle figure

| | | |
|-----|------------------------------------------------------------|----|
| 2.1 | Schermata di draw.io | 3 |
| 2.2 | Schermata di LoCoModeler | 4 |
| 2.3 | Schermata di TiVE | 5 |
| 2.4 | Switch per la selezione della modalità in drawSE | 6 |
| 3.1 | Esempio di sentenza visiva, da [3] | 9 |
| 4.1 | Diagramma ER | 14 |

Elenco delle tabelle

| | | |
|-----|--------------------------------------------------------------------------|----|
| 4.1 | Specifica di un linguaggio, nel particolare di un Albero | 11 |
| 4.2 | Specifica LCSD di un diagramma ER, costruita sulla specifica sintattica. | 13 |

Capitolo 1

Introduzione

La comunicazione fra individui avviene in svariati modi, ad esempio attraverso il linguaggio verbale ed il linguaggio visivo (o linguaggio visuale).

Un linguaggio visuale non è altro che una forma di comunicazione, detta comunicazione visuale, che fa uso di simboli grafici o immagini. Simboli grafici, immagini e mappe sono esempi di elementi utilizzati all'interno della comunicazione visiva (o comunicazione visuale) che necessitano di un contesto per essere descritte in modo naturale. Spesso quest'ultima risulta essere molto più immediata e di facile comprensione rispetto alla tradizionale comunicazione verbale coposta di lettere e parole.

In questa tesi presento **TiveJS**, un'estensione della piattaforma draw.io, che sfrutta simboli e definizioni sematiche per il riconoscimento dei linguaggi diagrammatici e la traduzione di questi in altri linguaggi.

1.1 Motivazioni

La piattaforma già esistente, LoCoMoTiVe, si basa su un meccanismo client-server.

Il client è formato dalla piattaforma draw.io, opportunamente modificata, per la creazione di sentenze visuali. Il server è stato implementato in Java utilizzando i servlet per il riconoscimento e la traduzione delle sentenze visuali. Il funzionamento è molto semplice, il client esegue una chiamata HTTP di tipo POST contenute al suo interno un grafo o un diagramma, creato attraverso l'utilizzo di simboli ad hoc, in formato XML. Una volta ricevuta la sentenza visuale, il server la interpreta applicando le definizioni per poi restituire la traduzione semantica oppure dei messaggi di errore.

Le motivazioni che ci hanno portato alla creazione di un nuovo tool sono varie: rendere l'applicazione più scalabile e più veloce limitando l'interazione con il server a semplici accessi a pagine statiche; l'aggiornamento di TiVe all'ultima versione di draw.io. Essen-

do il core di TiveJS scritto completamente in JavaScript ora si integra perfettamente con la piattaforma estesa e con la manipolazione del grafo. Le definizioni dei linguaggi ora sono definite in formato JSON rendendo ancora più alta l'interoperabilità dei sistemi.

1.2 Organizzazione della Tesi

Nel capitolo 2 illustrerò i lavori correlati al mio progetto di tesi. Nel capitolo 3 tratterò dei linguaggi visuali e dei loro componenti fondamentali. Nel capitolo 4 parlerò del Local Context e delle corrispondenti specifiche sintattiche e semantiche. Nel capitolo 5 introdurrò il risultato del mio lavoro di tesi, TiveJS e le sue funzioni, e illustrerò i dettagli dell'implementazione e le tecnologie usate. Nel capitolo 6 illustrerò vari casi d'utilizzo da me studiati. Nel capitolo 7, presenterò possibili sviluppi futuri dell'applicazione e le conclusioni.

Capitolo 2

Lavori Correlati

Il mio lavoro di tesi, essendo principalmente un porting ed una rivisitazione, si basa su strumenti precedentemente sviluppati. Gli strumenti in questione sono descritti nel dettaglio nei seguenti sottoparagrafi.

2.1 Draw.io e mxGraph

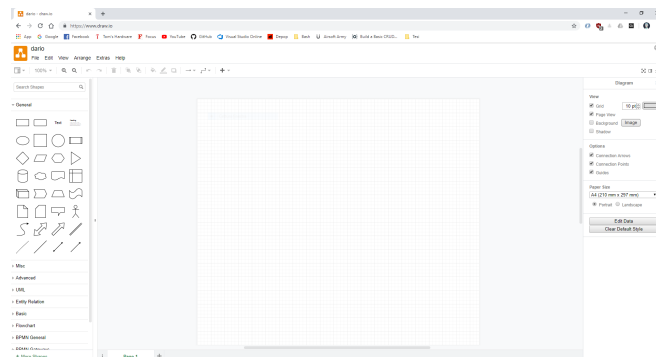


Figura 2.1 Schermata di draw.io

TiveJS, è basato su draw.io, un'applicazione web gratis che permette agli utenti di creare diagrammi e grafi direttamente dal proprio browser web, mostrato in figura 2.1. Ha un'integrazione con Google Drive e Dropbox per il salvataggio di dati che può avvenire anche con l'ausilio del localStorage del browser o attraverso il salvataggio di file sulla macchina. Draw.io è basato sulla libreria mxGraph. Il software è stato sviluppato nel 2005 dalla JGraph Ltd.

2.2 LoCoMoTiVE

All'attuale stato dell'arte troviamo l'ecosistema LoCoMoTiVE, ovvero un'unione di due software, LoCoModeler e Tive. Presentato in [4] e [5], questo tool permette l'analisi semantica basata sul contesto locale, vista nel dettaglio nel capitolo 4. Nei prossimi due paragrafi andrò ad illustrare singolarmente i due componenti di cui è composto.

2.2.1 LoCoModeler

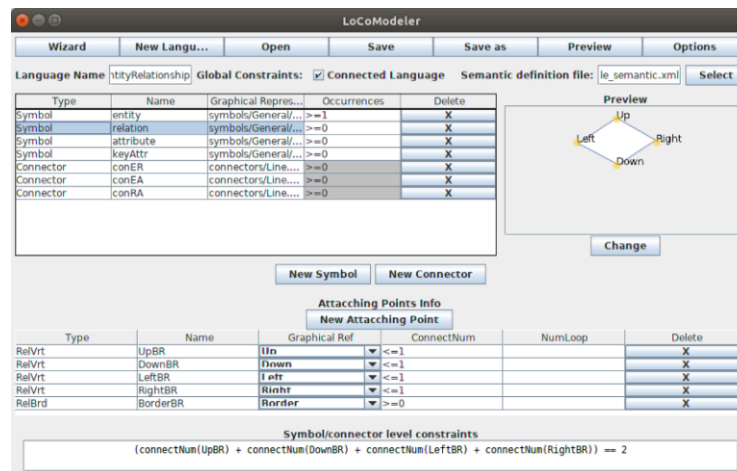


Figura 2.2 Schermata di LoCoModeler

Come descritto in [4], il modulo LoCoModeler consente ai designer la creazione e la modifica del linguaggio visivo in base al contesto locale, in maniera rapida e facile. Il suo output è la definizione in formato XML del linguaggio che verrà utilizzato durante il riconoscimento dei diagrammi. Una volta che il progettista ha completato la specifica del linguaggio, può compilarlo in un ambiente web (il modulo TiVE) per consentire agli utenti di disegnare frasi e verificarne la correttezza. Durante la definizione del linguaggio, questa funzione consente al progettista di controllare la correttezza delle specifiche.

Una schermata principale dell'interfaccia grafica del tool è mostrata nella Figura 2.2. Le sue componenti principali sono:

- Una casella di testo contenente il nome del linguaggio e una checkbox che sta ad indicare se il diagramma o grafo deve essere o non essere necessariamente connesso¹.

¹Un grafo è detto connesso se, per ogni coppia di vertici $(u, v) \in V$, esiste un cammino che collega u a v .

- Una tabella riportante le informazioni principali dei simboli e dei connettori inclusi nel linguaggio. E' possibile modificare o eliminare un elemento interagendo con la riga di questo. L'utente può aggiungere nuovi simboli o connettori usando i bottoni sottostanti la tabella.
- Un pannello (sulla destra) mostra un'anteprima grafica del simbolo o connettore selezionato nella tabella. E' possibile cambiare la rappresentazione grafica dell'elemento utilizzando il bottone *Change*.
- Una tabella (al centro) mostra le informazioni relative al simbolo o connettore selezionato. Ogni riga mostra un punto d'attacco e i relativi vincoli. E' possibile aggiungere nuove righe utilizzando i bottoni sovrastanti la tabella.
- Un'area di testo dove è possibile specificare i vincoli per il simbolo o il connettore attraverso espressioni simili al C².

La definizione di un nuovo linguaggio può avvenire grazie all'ausilio di un *Wizard* diviso in tre fasi.

2.2.2 TiVE

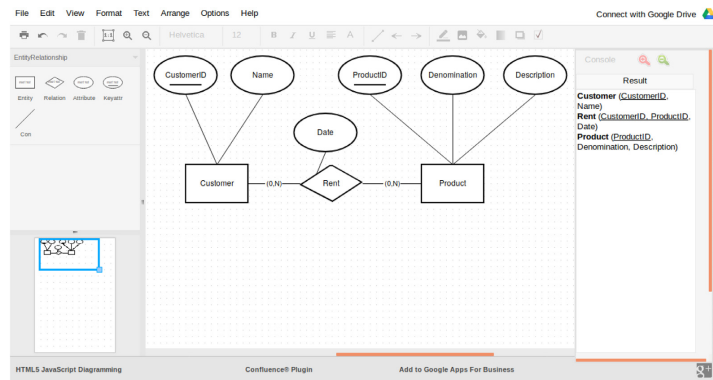


Figura 2.3 Schermata di TiVE

Una volta definito il linguaggio, i diagrammi possono essere composti utilizzando i simboli e i connettori definiti nella sua specifica. Questo può essere fatto attraverso un editor grafico TiVE, che è un'applicazione web che consente la composizione di diagrammi direttamente dal browser web.

Come mostrato nella Figura 2.3, l'applicazione è costituita da tre sezioni principali:

²Linguaggio di programmazione.

- Nella toolbar a sinistra troviamo la palette dei simboli e connettori utilizzabili per la creazione dei diagrammi.
- Nella zona centrale troviamo l'area di lavoro dove è possibile comporre i diagrammi trascinando gli elementi contenuti nella toolbar di sinistra.
- A destra troviamo la Console dove verrà mostrata la traduzione semantica o la lista di errori nel caso in cui si verificassero.

2.3 DrawSE

DrawSE è un'estensione di draw.io che permette la creazione di diagrammi con simboli altamente personalizzati. Le principali caratteristiche di drawSE sono le due modalità di editing: una per la creazione di simboli (*Shape Mode*) e una per la definizione dei punti d'attacco³ dei simboli (*AP Mode*). Le due modalità sono attivabili attraverso il selettore mostrato in figura 2.4



Figura 2.4 Switch per la selezione della modalità in drawSE

La *Shape Mode* permette la creazione di nuovi simboli e di personalizzarli in base al colore, lo spessore delle linee e così via. Un simbolo può essere formato anche dall'unione di più simboli semplici. Nella *AP Mode*, drawSE fornisce una palette con sette strumenti utili alla definizione degli *attaching point* del simbolo. Il punto d'attacco può essere composto da sette diverse forme geometriche:

- un punto
- una linea retta
- una linea curva
- un'area rettangolare
- un'area ellittica
- un contorno rettangolare
- un contorno ellittico

³Dove gli archi andranno ad attaccarsi sulla figura.

Altra funzionalità di drawSe è la creazione di set di simboli personalizzati o *custom palette*, ovvero un insieme dei simboli creati grazie alle due modalità specificate prima. Lo strumento è spiegato nel dettaglio in [2]. TiveJS fa uso delle palette generate in drawSE.

Capitolo 3

Linguaggi visuali

Nei precedenti capitoli ho parlato spesso di linguaggi verbali e linguaggi visuali e di quanto la comunicazione visuale può essere più efficiente della comunicazione verbale. In questo capitolo entrerò nel dettaglio senza dilungarmi andando ad illustrare quali sono le principali differenze tra un linguaggio visuale ed uno verbale, i componenti che lo compongono e in quali casi o contesti un linguaggio visivo è più efficace rispetto ad uno verbale.

3.1 Linguaggio verbale

Il linguaggio verbale è un gruppo di elementi, come suoni e parole, che messe insieme formano frasi e infine permettono la comunicazione fra individui. Da questo deriva la comunicazione verbale che è quindi costituita dalle parole usate quando parliamo o scriviamo.

3.2 Componenti

Ogni linguaggio è formato da un proprio insieme di componenti. Un linguaggio visuale si distingue principalmente dal linguaggio verbale per i componenti di cui è formato. Il linguaggio visivo si basa su simboli grafici o immagini. Elementi che il cervello umano interpreta e trasforma in concetti, linguaggio verbale ed emozioni. Quindi se il linguaggio visuale è costituito da testo e parole per la formazione di frasi, il linguaggio visivo è formato da simboli e disegni per formare sentenze visive. Una componente fondamentale di un linguaggio visuale è il contesto che viene dato ad ogni simbolo appartenente ad una frase visiva.

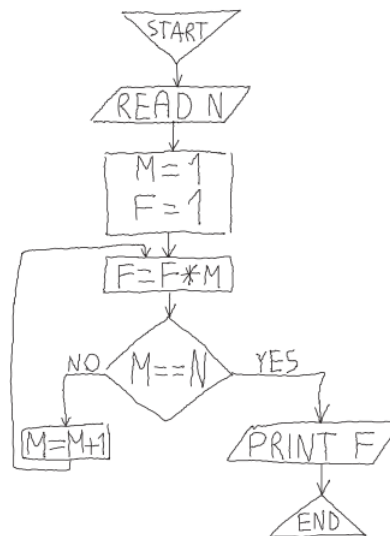


Figura 3.1 Esempio di sentenza visiva, da [3]

Ad esempio, come possiamo notare avvalendoci del disegno in figura 3.1, senza un contesto sono semplici simboli connessi fra loro da delle frecce. Invece, dando una definizione ai simboli del diagramma può essere interpretato come un diagramma di flusso o *Flowchart*¹.

3.3 Vantaggi

I vantaggi possono essere molteplici, innanzitutto un linguaggio visuale può essere molto più efficace e di facile comprensione rispetto al linguaggio verbale per via della sua semplicità e naturalità. Non ha lingue o convenzioni in quanto un disegno o un'immagine non dipende da lingue o standard.

¹Il diagramma di flusso (o *Flowchart*), in informatica, è una rappresentazione grafica delle operazioni da eseguire per l'esecuzione di un algoritmo.

Capitolo 4

Local Context

Il riconoscimento dei diagrammi viene effettuato in base al *Local Context*, presentato nei paper [3], [4] e [5].

Nel paper *Local context-based recognition of sketched diagrams* [3], il Local Context viene presentato come una nuova metodologia mirata alla creazione e all'implementazione di un framework¹ per il riconoscimento e l'interpretazione dei diagrammi. Nello specifico, i diagrammi possono contenere differenti elementi grafici quali simboli, connettori e testo. Una volta che i simboli sono stati identificati, il riconoscimento procede identificando il contesto locale di ogni simbolo. Il Local Context ha due diverse specifiche: sintattiche e semantiche.

4.1 Sintassi

Sempre in [3], viene definita la specifica sintattica di un linguaggio visuale. In particolare, usando il contesto locale, la specificazione della sintassi di un linguaggio visivo consiste di svariati elementi quali:

- Definizione dei simboli (token) che compongono il linguaggio visuale:
 - Definizione dell'apparenza "fisica" dei simboli (ad esempio forma, colore, ecc.);
 - Definizione degli attributi del simbolo della loro forma e del loro aspetto (ad esempio punti/area d'attaccamento, ecc);
 - Definizione dei vincoli locali al simbolo riguardanti gli attributi (ad esempio il numero di connessioni permesse ad un punto d'attaccamento, ecc.).

¹ 'Un framework, in generale, include software di supporto, librerie, un linguaggio per gli script e altri software che possono aiutare a mettere insieme le varie componenti di un progetto.', [1]

- Definizione delle relazioni/connettori e i loro vincoli locali;
- Dichiarazione di vincoli al livello del diagramma (ad esempio numero di occorrenze ammissibili di un simbolo, se i simboli e i connettori devono formare un grafo connesso);
- Una grammatica per definire ulteriori vincoli di sintassi del linguaggio (quando necessario).



| Token | Graphics | Token occurrences | name | Attachment points type | constraints |
|--------------------------------------------------|-----------------------------------------------------------------------------------|-------------------|------|------------------------|---------------------|
| ROOT |  | 1 | IN | enter | $connectNum = 0$ |
| | | | OUT | exit | $connectNum \leq 2$ |
| NODE |  | ≥ 0 | IN | enter | $connectNum = 1$ |
| | | | OUT | exit | $connectNum \leq 2$ |
| | | | | | |
| Connector | | | name | Attachment points type | constraints |
| POLYLINE | | | P0 | exit | $connectNum = 1$ |
| | | | P1 | enter | $connectNum = 1$ |
| Further constraint | | | | | |
| the spatial-relationship graph must be connected | | | | | |

Tabella 4.1 Specifica di un linguaggio, nel particolare di un Albero

Facendo riferimento alla tabella 4.1, ovvero la specifica di un linguaggio che identifica un Albero, possiamo notare che ogni riga identifica un simbolo (o token) e che ogni riga è composta da sei colonne, le prime tre sono indirizzate al livello del simbolo e le restanti tre colonne indicano i vincoli per gli attachment points:

- **Token:** indica il nome dell'elemento;
- **Graphics:** rappresentazione grafica dell'elemento e rappresentazione dei punti d'attaccamento evidenziati in giallo;
- **Token occurrences:** numero di occorrenze ammissibile per quel simbolo;
- **name:** può essere composta da più righe ed indica i nomi dei punti d'attaccamento;
- **type:** la tipologia di AP², un tipo di punto di attaccamento può avere più nomi;

²Punti d'attaccamento.

- **constraints:** i vincoli al livello dei punti d'attaccamento, *connectNum* indica il numero di archi incidenti a quel determinato AP.

I vincoli al livello della sentenza sono riportati nell'ultima riga. Una parte della formalizzazione della tabella 4.1 in formato XML è mostrata nello snippet di codice 4.1, l'intera specifica è consultabile nell'appendice A.1.

Listing 4.1 Frammento della definizione in formato XML di un linguaggio, nel particolare la specifica per il simbolo *root* (o radice).

```
<token name="root" ref="circle.svg" occurrences=="1">
  <ap type="enter" name="in" ref="hiSC" connectNum=="0"/>
  <ap type="exit" name="out" ref="lowSC" connectNum=="2"/>
</token>
```

Nel paper [4], viene esteso il concetto della specifica semantica esposta in [3] aggiungendo nuove caratteristiche alla specifica del contesto locale per permettere la specifica di linguaggi visuale più complessi quali diagrammi entità-relazione, use case diagrams e class diagram. In più viene presentato un tool (LoCoMoTiVE, specificato nel paragrafo 2.2) che implementa il framework *Local Context*.

Le principali caratteristiche aggiunte sono tre:

- La definizione di vincoli al livello del simbolo può coinvolgere più di un area di attaccamento di un simbolo/connettore al contrario di vincolare aree d'attaccamento individuali;
- La definizione di un vincolo per limitare auto-cicli di un connettore;
- Assegnazione di più tipi alle aree di attaccamento.

4.2 Semantica

In [5] viene esteso il framework definendo una nuova tecnica per una traduzione semantica del linguaggio visuale basata sul contesto locale. Questa tecnica usa delle espressioni simili a quelli di XPath, chiamate SGPath, illustrate nel dettaglio nel capitolo 5.

Per consentire la traduzione semantica di un linguaggio visuale è stata descritta una definizione semantica basata sul contesto locale o *LCSD*³ e il suo algoritmo per

³Local Context-based Semantic Definition

valutarla. L'LCSD consiste di una sequenza di regole semantiche una per ogni elemento del linguaggio. Ogni regola calcola una proprietà attraverso delle procedure che fanno uso degli SGPath o esegue un'azione che dipende dalla proprietà e gli attributi. Ogni proprietà può avere una post-condizione.

Attraverso le post-condizioni, una LCSD può dare una definizione migliore della struttura sintattica; attraverso le azioni restituisce una traduzione delle sentenze. Un'azione dipende da proprietà e attributi.

| ENTITY | | | | |
|--------------------------------------------------------------------------------------------------------------------------------------------------|-----------|--------------------------------------|--------------|-----------------|
| Property | Procedure | Params | | Post-condition |
| \$Key : list<string> | add | CON_E_A/KEY_ATTR | @KeyName | size(\$Key) > 0 |
| \$Attributes : list<string> | add | CON_E_A/ATTRIBUTE | @AttrName | |
| | addAll | CON_E_R[@Cardin='([01],1)']/RELATION | \$Attributes | |
| print(@EntName + " (<u>" + explode(" ", \$Key) + "</u>"); if(size(\$Attributes) > 0) print(" " + explode(" ", \$Attributes)); print("\n"); | | | | |

| RELATION | | | | |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------|------------------------------------|-----------|----------------------|
| Property | Procedure | Params | | Post-condition |
| \$Key : list<string> | addAll | CON_E_R[@Cardin='([01],N)']/ENTITY | \$Key | |
| \$Attributes : list<string> | add | CON_R_A/ATTRIBUTE | @AttrName | |
| \$Entities : list<string> | add | CON_E_R[@Cardin='([01],N)']/ENTITY | @EntName | |
| \$KeyAttrs : list<string> | add | CON_R_A/KEY_ATTR | @KeyName | size(\$KeyAttrs) = 0 |
| if(size(\$Entities) == 2) { print(@RelName + " (<u>" + explode(" ", \$Key)) + "</u>"); if(size(\$Attributes) > 0) print(" " + explode(" ", \$Attributes)); print("\n"); } | | | | |

Tabella 4.2 Specifica LCSD di un diagramma ER, costruita sulla specifica sintattica.

La tabella 4.2 mostra una specifica semantica di un diagramma ER⁴. Ogni tabella fornisce le regole semantiche di un elemento, in questo caso per i simboli ENTITY e RELATION, rispettivamente. Ogni riga della specifica è composta dai seguenti elementi:

- **Property:** indica il nome della proprietà (preceduta dal carattere \$) e il tipo.
- **Procedure:** indica il nome della procedura da utilizzare per assegnare/modificare il valore di una proprietà, ogni proprietà può avere più procedure.
- **Params:** i parametri da utilizzare nella procedura. Il primo parametro è un SGPath, il secondo parametro è il nome della proprietà dove leggere le informazioni del/i simbolo/i raggiungibile/i con quel percorso.
- **Post-condition:** la post-condizione che deve essere rispettata al termine dell'esecuzione della procedura.

⁴Entità Relazione.

L'ultima colonna indica l'azione da eseguire. A differenza dell'attuale implementazione, TiVeJS può eseguire anche azioni parziali, ovvero posizionata tra le proprietà e non dopo le proprietà.

In questo caso, applicando le regole semantiche definite nella tabella 4.2 al diagramma

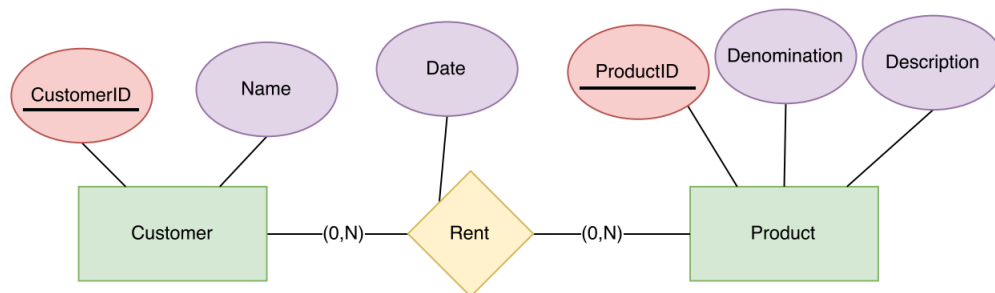


Figura 4.1 Diagramma ER

4.1 produrrà la seguente traduzione semantica:

Customer (CustomerID, Name)

Product (ProductID, Denomination, Description)

Rent (CustomerID, ProductID, Date)

Tutti i dettagli implementativi saranno illustrati nel prossimo capitolo.

Capitolo 5

TiveJS

In questo lavoro, come introdotto, presento TiveJS, che è un porting ed un'evoluzione di Tive e come esso è in grado di: (1) permettere la progettazione di sentenze visive grazie all'ausilio di palette di simboli personalizzate, (2) il riconoscimento di linguaggio diagrammatici sfruttando e (3) la traduzione di quest'ultimi.

5.1 Funzionamento

5.1.1 Progettazione sentenze visive

5.1.2 Riconoscimento grafo

5.1.3 Applicazione definizioni

5.1.4 Traduzione

5.2 Implementazione

5.2.1 Tecnologie usate

Capitolo 6

Contesti d'utilizzo

6.1 Entity Relationship

6.2 Flowchart

6.3 Tree

Capitolo 7

Conclusioni

7.1 Sviluppi futuri

Appendice A

Codici

Listing A.1 Specifica di un linguaggio in formato XML, in questo caso quello per Albero Binario

```
<language name="binaryTree">

  <token name="root" ref="circle.svg" occurrences=="1">
    <ap type="enter" name="in" ref="hiSC" connectNum=="0"/>
    <ap type="exit" name="out" ref="lowSC" connectNum=="2"/>
  </token>

  <token name="node" ref="circle.svg" occurrences=">0">
    <ap type="enter" name="in" ref="hiSC" connectNum=="1"/>
    <ap type="exit" name="out" ref="lowSC" connectNum=="2"/>
  </token>

  <connector ref="polyline">
    <cap type="exit" ref="p0" connectNum=="1" />
    <cap type="enter" ref="p1" connectNum=="1" />
  </connector>

  <constraint>connected</constraint>
</language>
```

Bibliografia

- [1] URL <http://www.pc-facile.com/glossario/framework/>.
- [2] cluelab. About - drawse. URL <https://cluelab.github.io/drawSE/>.
- [3] Vittorio Fuccella Gennaro Costagliola, Mattia De Rosa. Local context-based recognition of sketched diagrams. *Journal of Visual Languages and Computing*, pages 955–962, 10 2014. doi: 10.1016/j.jvlc.2014.10.021.
- [4] Vittorio Fuccella Gennaro Costagliola, Mattia De Rosa. Extending local context-based specifications of visual languages. *Journal of Visual Languages and Computing*, pages 184–195, 12 2015. doi: 10.1016/j.jvlc.2015.10.013.
- [5] Vittorio Fuccella Gennaro Costagliola, Mattia De Rosa. Using the local context for the definition and implementation of visual languages. *Computer Languages, Systems and Structures*, pages 20–38, 12 2018. doi: 10.1016/j.cl.2018.04.002.