WP2.2 - Algoritmi per l'analisi dei goal diagram

Università di Trento

Abstract.
This deliverable presents the results of the WP2-Task 2.2 and notably the development of algorithms for goal diagram analysis. We have defined and implemented algorithms that allow one to analyze in a diagram the satisfiability of goals on the base of the satisfiability of their subgoals and the goals and softgoals that can contribute to their satisfiability. Both qualitative and quantitative approaches have been pursued.

Document Identifier	Deliverable Dm.n
Project	MIUR-FIRB project RBNE0195K5 "Knowledge Level Automated Software Engineering"
Version	v1.0
Date	November 5, 2006
State	Final
Distribution	Public

Acknowledgements.

This document is part of a research project funded by the FIRB 2001 Programme of the "Ministero dell'Istruzione, dell'Università e della Ricerca" as project number RBNE0195K5.

The partners in this project are: Istituto Trentino di Cultura (Coordinator), Università degli Studi di Trento, Università degli Studi di Genova, Università degli Studi di Roma "La Sapienza", DeltaDator S.p.A..

Executive Summary

This deliverable presents the results of the WP2-Task 2.2 and notably the development of algorithms for goal diagram analysis. We have defined and implemented algorithms that allow one to analyze in a diagram the satisfiability of goals on the base of the satisfiability of their subgoals and the goals and softgoals that can contribute to their satisfiability. Both qualitative and quantitative approaches have been pursued.

In particular, in this deliverable we focus on goal-oriented requirements analysis and reasoning with goal models in forward and backward fashion. We present a semantic matching approach as well as its optimizations which is used in order to identify semantically related entities among schemas and goal diagrams. Finally, we present a CASE tool for security requirements engineering.

Contents

Chapter 1

Introduction

The main motivation for supporting early requirements [DvLF93a, Yu95] is to develop a rich conceptual framework for modeling and analyzing processes that involve multiple participants (both humans and software systems) and the intentions that these processes are supposed to fulfill. By doing so, one can relate the functional and non-functional requirements of the system-to-be to relevant stakeholders and their intentions. Tropos [ea06] adopts Eric Yu's *i** model [Yu95] which offers *actors* (*agents*, *roles*, or *positions*), *goals*, and *actor dependencies* as primitive concepts for models used in different phases of software development. In particular, Tropos is intended to support four phases of software development: *early requirements analysis*, concerned with the understanding of a problem by studying its organizational setting; *late requirements analysis*, where the system-to-be is described within its operational environment, along with relevant functions and qualities; *architectural design*, where the systems global architecture is defined in terms of subsystems, interconnected through data, control, and other dependencies; and *detailed design*, where behavior of each software component is defined in further detail.

In many papers that have advocated the use of the Tropos methodology, the goal analysis is given a prominent role. In a nutshell, software development begins by identifying relevant stakeholders (represented as actors) and their goals. These root goals are analyzed, refined, and delegated to existing on new actors. The system-to-be and its components come about as new actors who are responsible for the fulfillment of some of the original or refined goals. The whole process ends when sufficient goals have been delegated so that if all actors fulfill their responsibilities, all root goals are fulfilled. Thus, the objective of the first part of this deliverable is build on top of works in [GMS04, GNMS04, GNMS02] and to make this goal analysis process concrete by using a formal goal model developed in [GNMR03, RGM04]. This goal model supports both qualitative and quantitative relationships between goals, and can be used to perform two types of analysis. The first type (forward reasoning) answers questions of the form: Given a goal model, and assuming that certain leaf goals are fulfilled, are all root goals fulfilled as well? The second type of analysis (backward reasoning) solves problems of the form: Given a goal model, find a set of leaf goals that together fulfill all root goals.

Another important operation for goal diagrams is comparing them against other goal diagrams and existing design patterns [GSY05, GSY06, GY04, MSGPdS04, GSY04, GS03a]. Match is therefore a critical operator here, which is studied in this deliverable. The match operator takes two graph-like structures and produces a mapping between the nodes of the graphs that correspond semantically to each other.

Many diverse solutions of match have been proposed so far in the literature, e.g., [DLD$^+$04, KN03, DR02, MBR01, MGMR02, BCV99, BSZ03, GATTM05]. We focus on a schema-based solution, namely a matching system exploiting only the schema information, thus not considering instances. We follow a novel approach called *semantic matching* [GS03b]. This approach is based on two key ideas. The first is that we calculate mappings between schema elements by computing *semantic relations* (e.g., equivalence, more general, disjointness), instead of computing coefficients rating match quality in the [0,1] range, as it is the case in the most previous approaches, see, for example, [DR02, MBR01, MGMR02]. The second idea is that we determine semantic relations by analyzing the *meaning* (concepts, not labels) which is codified in the elements and the structures of schemas. In particular, labels at nodes, written in natural language, are translated into propositional formulas which explicitly codify the label's intended meaning. This allows us to translate the matching problem into a propositional validity problem, which can then be efficiently resolved using (sound and complete) state of the art propositional satisfiability (SAT) deciders, e.g., [Ber].

Software designers have recognized the need to integrate most non-functional requirements (such as reliability and performance) into the software development processes, but security still remains an afterthought. This often means that security mechanisms have to be fitted into a pre-existing design which may not be able to accommodate them due to potential conflicts with functional requirements or usability.

This has spurred a number of researchers to model security requirements into "standard" software engineering methodologies. The major limitation of many proposals is that they treat security in system-oriented terms. In other words, they are targeted to *model a computer system* and the policies and access control mechanisms it supports. Therefore, as a last theme of the deliverable we extend the Tropos goal analysis model to support security requirements and present a CASE tool for design and verification of functional and security requirements, the ST-Tool [GMMZ05, GMM$^+$05].

1.1 Technical contribution

We present a formal goal model defined and analyzed in [GNMR03, RGM04] and adopt it to make the goal analysis process concrete through the use of forward and backward reasoning for goal models. The formal goal analysis is illustrated through examples, using an implemented goal reasoning tool. Moreover we show an extension of the tool for modeling and analyzing functional and security requirements.

We discuss *match*, namely an operator that takes two graph-like structures (e.g., Goal Diagrams) and produces a mapping between the nodes of these graphs that correspond semantically to each other. In particular, we introduce basic and optimized algorithms for semantic schema matching, which realize the match operation, and we discuss their implementation within the S-Match system. We also validate the approach and evaluate S-Match against three state of the art matching systems.

Finally, we have designed and developed the ST-Tool to support the Secure Tropos methodology [GMMZ04]. The main goals of the tool include:

- Graphical environment: a visual framework to draw functional and security requirements;
- Formalization: support to translate models into formal specifications;
- Analysis capability: a front-end to external tools for formal analysis.

1.2 Plan of the deliverable

The rest of the deliverable is structured as follows. The second chapter focuses on goal-oriented requirements analysis and reasoning with goal models in forward and backward fashion. The third chapter is devoted to a semantic matching approach as well as its optimizations which is used in order to identify semantically related entities of schemas and goal diagrams under consideration. Finally, the last chapter discusses a CASE tool for security requirements engineering.

Chapter 2

Goal-Oriented Requirements Analysis and Reasoning in the Tropos Methodology

2.1 A Case Study

To make the analysis more concrete we start with a case study. It is a revised version of the case study presented in [CKM02]. *Media Shop* is a store selling and shipping different kinds of media items such as books, newspapers, magazines, audio CDs, videotapes, and the like. *Media Shop* customers (on-site or remote) can use a periodically updated catalogue describing available media items to specify their order. *Media Shop* is supplied with the latest releases from *Media Producer* and in-catalogue items by *Media Supplier*. To increase market share, *Media Shop* has decided to open up a B2C retail sales front on the internet. With the new setup, a customer can order *Media Shop* items in person, by phone, or through the internet. The system has been named *Medi@* and is available on the world-wide-web using communication facilities provided by *Telecom Cpy*. It also uses financial services supplied by *Bank Cpy*, which specializes on on-line transactions. The basic objective for the new system is to allow an on-line customer to examine the items in the *Medi@* internet catalogue, and place orders.

There are no registration restrictions, or identification procedures for *Medi@* users. Potential customers can search the on-line store by either browsing the catalogue or querying the item database. The catalogue groups media items of the same type into (sub)hierarchies and genres (e.g., audio CDs are classified into pop, rock, jazz, opera, world, classical music, soundtrack, . . .) so that customers can browse only (sub)categories of interest. An on-line search engine allows customers with particular items in mind to search title, author/artist and description fields through keywords or full-text search. If the item is not available in the catalogue, the customer has the option of asking *Media Shop* to order it, provided the customer has editor/publisher references (e.g., ISBN, ISSN), and identifies herself (in terms of name and credit card number). Details about media items include title, media category (e.g., book) and genre (e.g., science-fiction), author/artist, short description, editor/publisher international references and information, date, cost, and sometimes pictures (when available).

An actor diagram is a graph involving *actors* who have *strategic dependencies* among each other. A dependency represents an "agreement" (called *dependum*) between two actors: the *depender* and the *dependee*. The *depender* depends on the *dependee,* to deliver on the dependum. The dependum can be a *goal* to be fulfilled, a *task* to be performed, or a *resource* to be delivered. In addition, the depender may depend on the dependee for a *softgoal* to be fulfilled. Softgoals represent vaguely defined goals, with no clear-cut criteria for their fulfillment. Graphically, actors are represented as circles; dependums – goals, softgoals, tasks and resources – are respectively represented as ovals, clouds, hexagons and rectangles; and dependencies have the form *depender* → *dependum* → *dependee*.

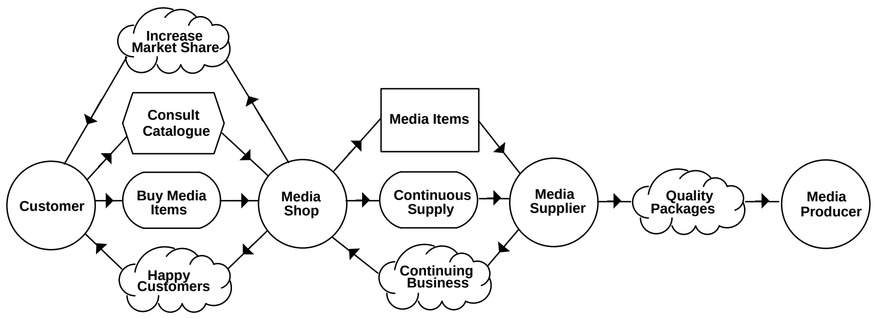


Figure 2.1: Actor diagram for a Media Shop

Figure 2.1 depicts the actor diagram for our Medi@ example. The main actors are *Customer*, *Media Shop*, *Media Supplier* and *Media Producer*. *Customer* depends on *Media Shop* to fulfill her goal: *Buy Media Items*. Conversely, *Media Shop* depends on *Customer* to *increase market share* and make "*customers happy*". Since the dependum *Happy Customers* cannot be defined precisely, it is represented as a softgoal. The *Customer* also depends on *Media Shop* to *consult the catalogue* (task dependency). Furthermore, *Media Shop* depends on *Media Supplier* to supply media items in a continuous way and get a *Media Item* (resource dependency). The items are expected to be of good quality because, otherwise, the *Continuing Business* dependency would not be fulfilled. Finally, *Media Producer* is expected to provide *Media Supplier* with *Quality Packages.*

Actor diagrams are extended during early requirements analysis by incrementally adding more specific actor dependencies which come out from a means-ends analysis of each goal. This analysis is specified using rationale diagrams.

A rationale diagram appears as a balloon within which goals of a specific actor are analyzed and dependencies with other actors are established. Goals are decomposed into subgoals and positive/negative contributions of subgoals to goals are specified. The intuitive meaning of the positive (+ and ++) and negative (– and – –) contributions, is that the satisfaction of a goal G contributes positively (negatively) to the satisfaction (denial) of another goal G'. + and ++ (– and – –) specify the different strength of the contribution. In the next section such relationships are formally defined.

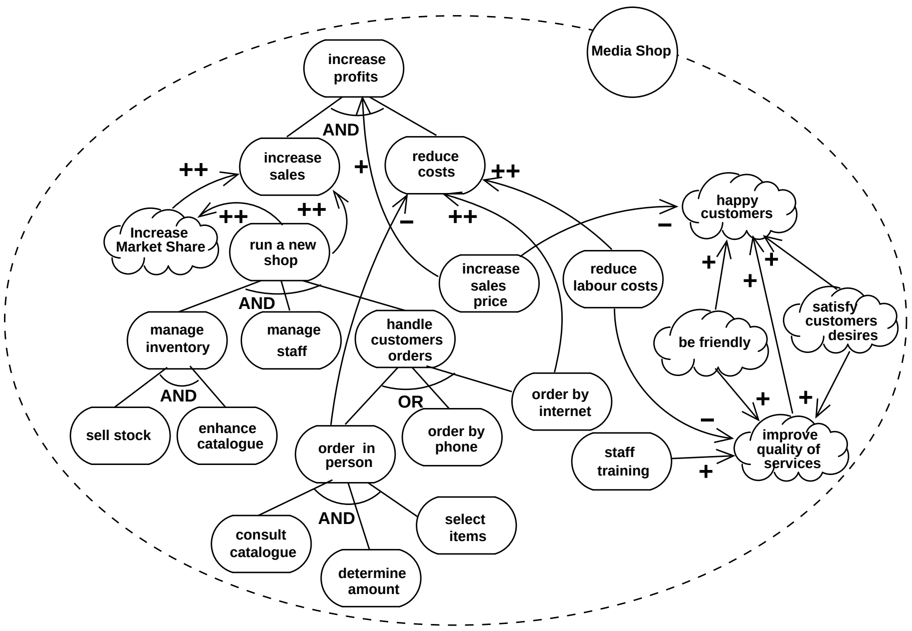


Figure 2.2: Rationale diagram for the *Media Shop*

Figure 2.2 shows the rationale diagram for the *Media Shop* actor focusing on the goal *increase profits*, which is and-decomposed in *increase sales* and *reduce costs*. The *Media Shop* has also the softgoals *happy customers*, *increase market share*, and *improve quality of services*. The goal *run a new shop* gives a positive (++) contribution to the goal *increase sales* and to the softgoal *increase market share*. in order to satisfy to goal *run a new shop*, the *Media Shop* has to *manage inventory*, *manage staff*, and *handle customers orders*. these goals are further refined, so for instance *handle customers orders* can be achieved in three different ways: *order in person*, *order by phone*, or *order by Internet*. Each of these goals gives a different contribution to the goal *reduce costs*; namely, a positive contribution (++) for *order in person* and a negative contribution (–) for *order by internet*. The goal *reduce labour costs* contributes negatively to *improve the quality if services*, while it gives a positive contribution to the goal *reduce costs*. Finally, the softgoal *happy customers* receives positive contributions from the softgoals *be friendly*, *satisfy customers desires*, and *improve quality of service*.

Late Requirements Analysis

During *late requirements* analysis, the conceptual model developed during early requirements is extended to include the system-to-be as a new actor, along with dependencies between this actor and others in its environment. These dependencies define functional (goals) and non-functional (softgoals) requirements for the system-to-be. Actor diagrams and rationale diagrams are used also in this phase.

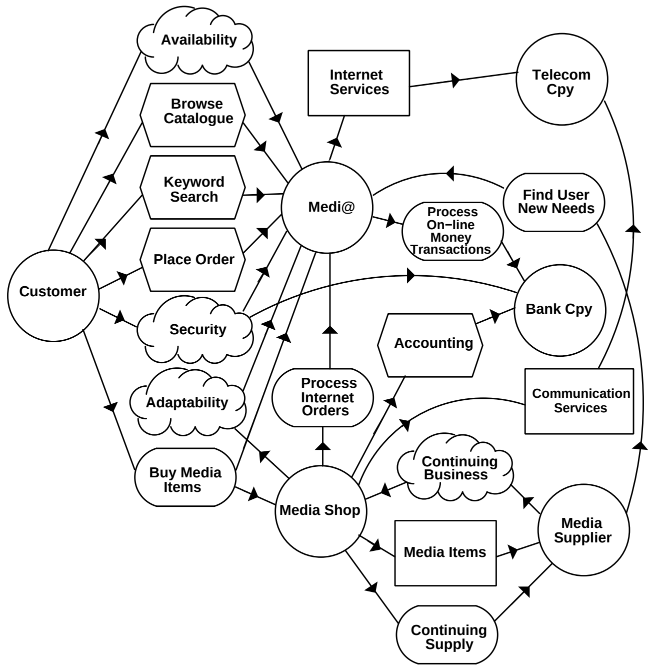


Figure 2.3: Actor diagram for the Media Shop

For our example, the *Medi@* system is viewed as a full-fledge actor in the strategic dependency model depicted in Figure 2.3. With respect to the actors previously identified, *Customer* depends on *Media Shop* to buy media items while *Media Shop* depends on *Customer* to increase market share and make them happy (with *Media Shop* service). *Media Supplier* is expected to supply *Media Shop* with media items in a continuous way since depending on the latter for continuing business. It can also use *Medi@* to determine new needs from customers, such as media items not available in the catalogue while expecting *Media Producer* to provide her with *quality packages*. As indicated earlier, *Media Shop* depends on *Medi@* for processing internet orders and on *Bank Cpy* to process business transactions. *Customer*, in turn, depends on *Medi@* to place orders through the internet, to search the database for keywords, or simply to browse the on-line catalogue. With respect to relevant qualities, *Customer* requires that transaction services be secure and available, while *Media Shop* expects *Medi@* to be easily adaptable (e.g., catalogue enhancing, item database evolution, user interface update, . . .). Finally, *Medi@* relies on internet services provided by *Telecom Cpy* and on secure on-line financial transactions handled by *Bank Cpy*.

Although a strategic dependency model provides hints about why processes are structured in a certain way, it does not sufficiently support the process of suggesting, exploring, and evaluating alternative solutions. As late requirements analysis proceeds, *Medi@*

is given additional responsibilities, and ends up as the depender of several dependencies. Moreover, the system is decomposed into several sub-actors which take on some of these responsibilities. This decomposition and responsibility assignment is realized using the same kind of means-ends analysis along with the strategic rationale analysis illustrated in Figure 2.2. Hence, the analysis in Figure 2.4 focuses on the system itself, instead of an external stakeholder.

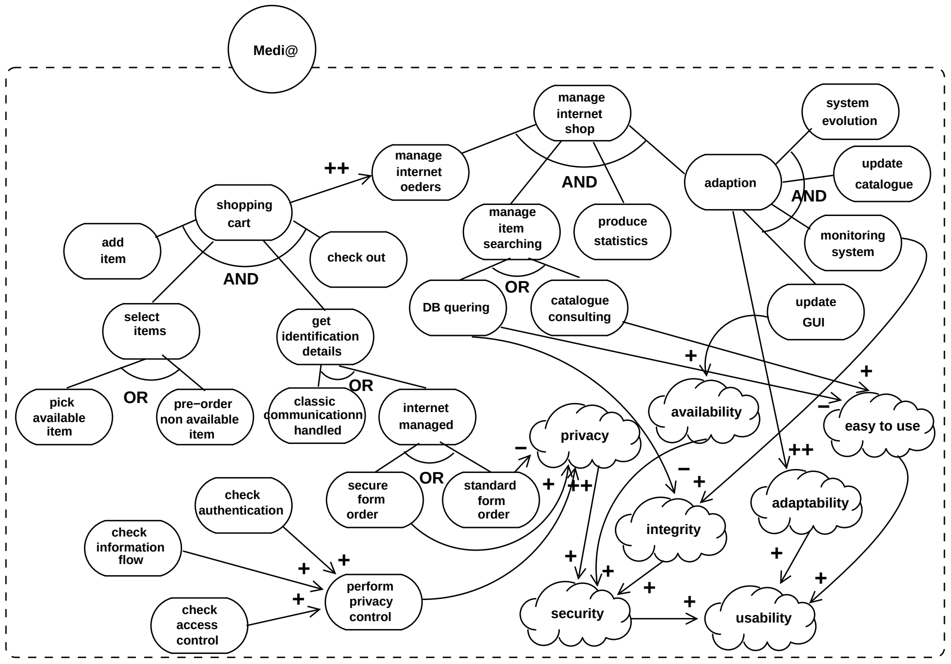


Figure 2.4: Actor diagram for the Media Shop focusing on the goal *internet shop managed*

The figures shows the analysis for the goal *manage internet shop*, which solves partially the dependencies with the other actors reported in Figure 2.3. The goal is firstly refined into goals *manage internet order*, *manage item searching*, *produce statistics* and *adaptation*. To achieve *manage internet order* is used the goal *shopping cart* which is decomposed into subgoals *select item*, *add item*, *check out,* and *get identification details*. These are the main process activities required to design an operational on-line shopping cart [Con00]. The latter (goal) is achieved either through subgoal *classic communication handled* dealing with phone and fax orders or *internet handled* managing secure or standard form orderings. To allow for the ordering of new items not listed in the catalogue, *select item* is also further refined into two alternative subgoals, one dedicated to select catalogued items, the other to pre-order unavailable products. To provide sufficient support (++) to the *adaptability* softgoal, *adaptation* is refined into four subgoals dealing with catalogue updates, system evolution, interface updates and system monitoring. The goal *manage item searching* might alternatively be fulfilled through goals *DB querying* or *catalogue consulting* with respect to customers' navigating desiderata, i.e., searching with particular items in mind by using search functions or simply browsing the catalogued products.

The figure reports also the analysis for the softgoals *security* and *usability*. *Security* receive positive contribution from the satisfaction of softgoals *privacy*, *availability*, and *integrity*, whereas *usability* from *adaptability* and *easy to use*. Notice, that *standard form order* gives a negative contribution to the *privacy*. Of course the analysis should include other non-function requirements, but for sake of simplicity we just focus on these two.

2.2 Goal Models

The concept of goal has been used in different areas of Computer Science since the early days of the discipline. In AI, problem solving and planning systems have used the notion of goal to describe desirable states of the world [NS]. More recently, goals have been used in Software Engineering [vL00, Rol03] to model early requirements [DvLF93b] and non-functional requirements [MCN92] for a software system.

Unfortunately, the approaches presented in the literature for modeling and analyzing goals do not work for many domains where goals can't be formally defined, and the relationships among them can't be captured by semantically well-defined relations such as AND/OR ones. For example, in our example the goal "*happy customers*" has no formally defined predicate which prescribes its meaning, though one may want to define necessary conditions for such a goal to be satisfied. Moreover, such a goal may be related to other goals, such as "*improve quality of service*" and "*be friendly*", in the sense that the latter obviously contribute to the satisfaction of the former, but this contribution is partial and qualitative. In other words, if the latter goals are satisfied, they certainly contribute towards the satisfaction of the former goal, but certainly do not guarantee it.

In this section we present the formal model for goals adopted in Tropos, which allows the software engineer to cope with qualitative relationships and inconsistencies among goals. In particular, in following two sections we present the notions of goal graphs and the axiomatic representation of goal relations.

2.2.1 Goal Graphs

We consider sets of goal nodes G_i and of relations $(G_1, ..., G_n) \xmapsto{r} G$ over them, including the $(n+1)$-ary relations and, or and the binary relations $+_S$, $-_S$, $+_D$, $-_D$, $++_S$, $--_S$, $++_D$, $--_D$, $+$, $-$, $++$, $--$. We briefly recall the intuitive meaning of these relations:

- $(G_1, ..., G_i, ...G_n) \xmapsto{and} G$ means that G is satisfied [resp denied] if all $G_1, ..., G_n$ are satisfied [resp. if at least one G_i is denied];
- $(G_1, ..., G_i, ...G_n) \xmapsto{or} G$ means that G is denied [resp satisfied] if all $G_1, ..., G_n$ are denied [resp. if at least one G_i is satisfied];

- $G_2 \xmapsto{+_S} G_1$ [resp. $G_2 \xmapsto{++_S} G_1$] means that if G_2 is satisfied, then there is some [resp. a full] evidence that G_1 is satisfied, but if G_2 is denied, then nothing is said about the denial of G_1;

- $G_2 \xmapsto{-_S} G_1$ [resp. $G_2 \xmapsto{--_S} G_1$] means that if G_2 is satisfied, then there is some [resp. a full] evidence that G_1 is denied, but if G_2 is denied, then nothing is said about the satisfaction of G_1.

- $G_2 \xmapsto{-_D} G_1$ [resp. $G_2 \xmapsto{--_D} G_1$] means that if G_2 is denied, then there is some [resp. a full] evidence that G_1 is satisfied, but if G_2 is satisfied, then nothing is said about the denial of G_1;

- $G_2 \xmapsto{+_D} G_1$ [resp. $G_2 \xmapsto{++_D} G_1$] means that if G_2 is denied, then there is some [resp. a full] evidence that G_1 is denied, but if G_2 is satisfied, then nothing is said about the satisfaction of G_1.

The names $+_S$, $-_S$, $+_D$, $-_D$, $++_S$, $--_S$, $++_D$, $--_D$ have the following intuitive meaning: the "S" [resp. "D"] symbol denotes the fact that the satisfiability [resp. deniability] value of the source goal is propagated; the "+" [resp. "-"] symbol denotes the fact that the propagation is positive [resp. negative], in the sense that satisfiability propagates to satisfiability [resp. deniability] and deniability propagates to deniability [resp. satisfiability].

The meaning of or, $+_D$, $-_D$, $++_D$, $--_D$ is dual w.r.t. and, $+_S$, $-_S$, $++_S$, $--_S$ respectively. (By "dual" we mean that we invert satisfiability with deniability.) The relations $+$, $-$, $++$, $--$ are defined such that each $G_2 \xmapsto{r} G_1$ is a shorthand for the combination of the two corresponding relationships $G_2 \xmapsto{r_S} G_1$ and $G_2 \xmapsto{r_D} G_1$. (We call the first kind of relations *symmetric* and the latter two *asymmetric*.) E.g., $G_2 \xmapsto{+} G_1$ is a shorthand for the combination of $G_2 \xmapsto{+_S} G_1$ and $G_2 \xmapsto{+_D} G_1$.

If $(G_1, ..., G_n) \xmapsto{r} G$ is a goal relation we call $G_1 ... G_n$ the *source goals* and G the *target goal* of r, and we say that r is an *incoming relation* for G and an *outcoming relation* for G_1,...,G_n. Notice that all relations are *directional*, from the sources to the target goals. We call *boolean relations* the and and or relations, *partial contribution relations* the $+$ and $-$ relations and their asymmetric versions, *full contribution relations* $++$ and $--$ relations and their asymmetric versions. We call a *root goal* any goal with an incoming boolean relation and no outcoming ones, we call a *leaf goal* any goal with no incoming boolean relations.

We call a *path from* G_1 *to* G_k a sequence of goals $\pi := G_1, G_2, ..., G_k$ s.t., for every $i \in \{1, ..., k-1\}$, G_i and G_{i+1} are respectively a source goal and the target goal of some relation r_i. We call a *loop* a path from a goal to itself. We call a *diamond* a pair of paths $\langle \pi_1, \pi_2 \rangle$ both from G_1 to G_k if π_1 and π_2 contain no common goal except G_1 and G_k.

We call a *goal graph* a pair $\langle \mathcal{G}, \mathcal{R} \rangle$ where $\mathcal{G}$ is a set of goal nodes and $\mathcal{R}$ is a set of goal relations, subject to the following restrictions:

each goal has at most one incoming boolean relation; (2.1)

every loop contains at least one non-boolean relation arc. (2.2)

In practice, a goal graph can be seen as a forest of and/or trees whose nodes are connected by contribution relations arcs. Root goals are roots of and/or trees, whilst leaf goals are either leaves or nodes which are not part of the trees.

The presence of contribution relations makes the tasks of formal reasoning on goal graphs much less straightforward than in the case of simple AND/OR graphs. The following factors contribute to complicate the picture.

Asymmetric value propagation. The way satisfiability and deniability values are propagated may be *asymmetric*. For instance, the relation $G_2 \overset{++D}{\longmapsto} G_1$ suggests that the achievement of the goal G_2 is a necessary but not sufficient condition for achieving the goal G_1. In fact, if G_2 is denied, then there is a full evidence that G_1 is denied, but if G_2 is satisfied, then nothing is said about the satisfaction of G_1.

Partial evidence. The contribution relations described above may propagate only a *partial* evidence about the satisfiability/deniability of the target goals. This means that a formal semantics for goal graphs must provide *partial* satisfiability/deniability values for the goals, and provide rules for propagating both full and partial satisfiability/deniability values through the relations.

Conflicts. Different goals can provide contradictory contributions to the same goals. For instance, if the graph contains $G_1 \overset{+S}{\longmapsto} G$ and $G_2 \overset{-S}{\longmapsto} G$ and both G_1 and G_2 are satisfied, then the first relation induces some evidence that G is satisfied, whilst the second induces some evidence that G is denied. We call these situations, *conflicts*. To this extent, it is important to keep track of both satisfiability and deniability values for all goals.

Diamonds. The value of one goal alone can provide contradictory contributions to another goal due to the presence of diamonds. For instance, if the graph contains $(G_1, G_5) \overset{or}{\longmapsto} G_2$, $G_2 \overset{+S}{\longmapsto} G_4$, $G_1 \overset{--S}{\longmapsto} G_3$ and $G_3 \overset{+D}{\longmapsto} G_4$, and both G_1 and G_5 are satisfied, then the satisfiability of G_1 propagates to G_4 through the diamond $\langle G_1G_2G_4, G_1G_3G_4 \rangle$, providing both some evidence that G_4 is satisfied (path $G_1G_2G_4$) and some evidence that G_4 is denied (path $G_1G_3G_4$).

Loops. The satisfiability/deniability of one goal can provide a contribution contradicting itself due to the presence of loops. This is the typical situation in models containing negative feedback loops (see, e.g, the real-world example in [GNMR02]). For instance, if the graph contains $G_1 \overset{+}{\longmapsto} G_2$ and $G_2 \overset{-}{\longmapsto} G_1$, and if G_1 is satisfiable, then the fact that G_1 is satisfied propagates through G_2 providing some evidence that G_1 is denied.

2.2.2 Axiomatization of Goal Relationships

Let $G_1, G_2, ...$ denote goal labels. We introduce four distinct predicates over goals, FS(G), FD(G) and PS(G), PD(G), meaning respectively that there is (at least) *full* evidence that goal G is satisfied and that G is denied, and that there is at least *partial* evidence that G is satisfied and that G is denied. We also use the proposition $\top$ to represent the (trivially true) statement that there is at least a null evidence that the goal G is satisfied (or denied). Notice that the predicates state that there is *at least* a given level of evidence, because in a goal graph there may be multiple sources of evidence for the satisfaction/denial of a goal. We introduce a total order $\mathsf{FS}(G) \geq \mathsf{PS}(G) \geq \top$ and $\mathsf{FD}(G) \geq \mathsf{PD}(G) \geq \top$, with the intended meaning that $x \geq y$ if and only if $x \rightarrow y$. We call FS, PS, FD and PD the possible *values* for a goal.

We want to allow the deduction of *positive* ground assertions of type FS(G), FD(G), PS(G) and PD(G) over the goal constants of a goal graph. We refer to externally provided assertions as *initial conditions*. To formalize the propagation of satisfiability and deniability evidence through a goal graph $\langle \mathcal{G}, \mathcal{R} \rangle$, we introduce the axioms described in Figure 2.5. For instance, (2.3) state that full satisfiability and deniability imply partial

Goal	**Invariant Axioms**	
G :	$\mathsf{FS}(G) \rightarrow \mathsf{PS}(G), \quad \mathsf{FD}(G) \rightarrow \mathsf{PD}(G)$	(2.3)
Goal relation	**Relation Axioms**	
$(G_1, ..., G_i, ...G_n) \xmapsto{and} G$:	$(\bigwedge_i \mathsf{FS}(G_i)) \rightarrow \mathsf{FS}(G), \ (\bigwedge_i \mathsf{PS}(G_i)) \rightarrow \mathsf{PS}(G)$	(2.4)
	$\bigwedge_i (\mathsf{FD}(G_i) \rightarrow \mathsf{FD}(G)), \ \bigwedge_i (\mathsf{PD}(G_i) \rightarrow \mathsf{PD}(G))$	(2.5)
$(G_1, ..., G_i, ...G_n) \xmapsto{or} G$:	$(\bigwedge_i \mathsf{FD}(G_i)) \rightarrow \mathsf{FD}(G), \ (\bigwedge_i \mathsf{PD}(G_i)) \rightarrow \mathsf{PD}(G)$	(2.6)
	$\bigwedge_i (\mathsf{FS}(G_i) \rightarrow \mathsf{FS}(G)), \ \bigwedge_i (\mathsf{PS}(G_i) \rightarrow \mathsf{PS}(G))$	(2.7)
$G_2 \xmapsto{+S} G_1$:	$\mathsf{PS}(G_2) \rightarrow \mathsf{PS}(G_1)$	(2.8)
$G_2 \xmapsto{-S} G_1$:	$\mathsf{PS}(G_2) \rightarrow \mathsf{PD}(G_1)$	(2.9)
$G_2 \xmapsto{++S} G_1$:	$\mathsf{FS}(G_2) \rightarrow \mathsf{FS}(G_1), \ \mathsf{PS}(G_2) \rightarrow \mathsf{PS}(G_1)$	(2.10)
$G_2 \xmapsto{--S} G_1$:	$\mathsf{FS}(G_2) \rightarrow \mathsf{FD}(G_1), \ \mathsf{PS}(G_2) \rightarrow \mathsf{PD}(G_1)$	(2.11)
$G_2 \xmapsto{+D} G_1$:	$\mathsf{PD}(G_2) \rightarrow \mathsf{PD}(G_1)$	(2.12)
$G_2 \xmapsto{-D} G_1$:	$\mathsf{PD}(G_2) \rightarrow \mathsf{PS}(G_1)$	(2.13)
$G_2 \xmapsto{++D} G_1$:	$\mathsf{FD}(G_2) \rightarrow \mathsf{FD}(G_1), \ \mathsf{PD}(G_2) \rightarrow \mathsf{PD}(G_1)$	(2.14)
$G_2 \xmapsto{--D} G_1$:	$\mathsf{FD}(G_2) \rightarrow \mathsf{FS}(G_1), \ \mathsf{PD}(G_2) \rightarrow \mathsf{PS}(G_1)$	(2.15)

Figure 2.5: Ground axioms for the invariants and the propagation rules.

satisfiability and deniability respectively; for an and relation, (2.4) show that the full and partial satisfiability of the target node require respectively the full and partial satisfiability of all the source nodes; for a "$+_S$" relation, (2.8) show that only the partial satisfiability (but not the full satisfiability) propagates through a "$+_S$" relation. Thus, e.g., an and relation propagates the minimum satisfiability value (and the maximum deniability one), while a "$+_S$" relation propagates at most a partial satisfiability value. To this extent, a "$+_S$" relation can be seen as an and relation with an unknown partially satisfiable goal. Similar considerations hold for the other relations.

Notice that, combining (2.3) with (2.4), and (2.3) with (2.8), we have, respectively,

$$(G_2, G_3) \xmapsto{and} G_1 : \qquad (\mathsf{FS}(G_2) \wedge \mathsf{PS}(G_3)) \rightarrow \mathsf{PS}(G_1) \tag{2.16}$$

$$G_2 \xmapsto{+_S} G_1 : \qquad \mathsf{FS}(G_2) \rightarrow \mathsf{PS}(G_1). \tag{2.17}$$

To this extent, henceforth we implicitly assume that axioms (2.3) are always implicitly applied whenever possible. Thus, e.g., we say that $\mathsf{PS}(G_1)$ is deduced from $\mathsf{FS}(G_2)$ and $\mathsf{PS}(G_3)$ by applying (2.4) —meaning "applying (2.3) and then (2.4)" — or that $\mathsf{PS}(G_1)$ is deduced from $\mathsf{FS}(G_2)$ and $\mathsf{FS}(G_3)$ by applying (2.4) —meaning "applying (2.4) and then (2.3)".

Let $A : (\bigwedge_{i=1}^{n} v_i) \rightarrow v$ be a generic relation axiom for the relation r. We call the values v_i the *prerequisites values* and v the *consequence value* of axiom A, and we say that the values v_i are the *prerequisites* for v through r and that v is the *consequence* of the values v_i through r.

We say that an atomic proposition of the form FS(G), FD(G), PS(G) and PD(G) *holds* if either it is an initial condition or it can be deduced via modus ponens from the initial conditions and the ground axioms of Figure 2.5. We assume conventionally that $\top$ always holds. Notice that all the formulas in the framework described so far are propositional Horn clauses, so that deciding if a ground assertion holds not only is decidable, but also it can be decided in polynomial time.

A *weak conflict* holds if $(\mathsf{PS}(G) \wedge \mathsf{PD}(G))$, a *medium conflict* holds if either $(\mathsf{FS}(G) \wedge \mathsf{PD}(G))$ or $(\mathsf{PS}(G) \wedge \mathsf{FD}(G))$, while a *strong conflict* holds if $(\mathsf{FS}(G) \wedge \mathsf{FD}(G))$, for some goal G.

2.3 Reasoning with Goal Models

In this section we present two forms of reasoning with goal models, *forward* and *backward reasoning*, and we show how they are applied in Tropos.

2.3.1 Forward Reasoning

Given a goal graph and an initial values assignment to some goals, *input goals* from now on (typically leaf goals), forward reasoning focuses on the forward propagation of these initial values to all other goals of the graph according to the rules described in Section 2.2. Initial values represent the evidence available about the satisfaction and the denial of a specific goal, namely evidence about the state of the goal. Usually such a evidence corresponds to qualitative values of satisfaction or denial of a goal. This is mainly because the evidence is usually provided very vaguely by the stakeholders, during the interviews with the analyst, or elaborated from documents or other available sources of information.

For each goal we consider three values representing the current evidence of satisfiability and deniability of goal: F (full), P (partial), N (none). We admit also conflicting situations in which we have both evidence for satisfaction and denial of a goal. So for instance, we may have that for goal G we have fully (F) evidence for the satisfaction and at the same time partial (P) evidence for denial. This could represent a situation in which we have two difference sources of information that provide conflicting evidence, or a multiple decompositions of goal G, where some decompositions suggest satisfaction of G while others suggest denial.

After the forward propagation of the initial values, the user can look the final values of the goals of interest, *target goals* from now on (typically root goals), and reveal possible conflicts. In other words, the user observes the effects of the initial values over the goals of interests.

In [GNMR02, GNMR03] we have presented the algorithm for the forward propagation and we have shown soundness and completeness with respect to the axiomatization. In the algorithm, to each node G of the graph $\mathcal{G}$ we associate two variables $\mathsf{Sat}(G), \mathsf{Den}(G)$ ranging in $\{F, P, N\}$ (full, partial, none) such that $F > P > N$, representing the current evidence of satisfiability and deniability of goal G. For example, $\mathsf{Sat}(G_i) \geq P$ states that there is at least partial evidence that G_i is satisfiable. (To this extent, e.g., $\mathsf{Sat}(G_i) \geq P$ is equivalent to say that $\mathsf{PS}(G_i)$ holds, and so on.) As the goal graph may be cyclic, the process stops when a fixpoint is reached. Starting from assigning an initial set of input values for $\mathsf{Sat}(G_i)$, $\mathsf{Den}(G_i)$ to (a subset of) the goals in $\mathcal{G}$, we propagate the values through the goal relations.

Let us consider for instance the rationale diagram for the *Media Shop* presented in Figure 2.2. Let us suppose that we have full evidence for the satisfaction of goals *increase sales*, *increase sales price*, and *reduce labour costs*. The result of the forward propagation of these values is that we have full evidence for the satisfaction of the top goal *increase profits* and partial evidence for the denial of softgoals *happy customers* and *improve quality of services*.

As we have seen previously, the Tropos methodology analyzes the requirements of the system-to-be in terms of goal models. Goals basically represent the functional requirements, while the softgoals represent the non-functional requirements of the system. In the goal models, OR relationships are used to model possible alternatives, and the adoption of each of them can have a different impact on the satisfaction of the softgoals. So for instance, in Figure 2.4 the two alternatives *secure form order* and *standard order form* contribute respectively positively and negatively to the satisfaction of the softgoal *privacy*.

Forward reasoning is adopted in Tropos for evaluating the impact of the adoption of the different alternatives with respect to the softgoals of the system-to-be. Table 2.1 reports the results of the forward reasoning in four different situations for the goal model presented in Figure 2.4. The table shows only the results for the goals involved in OR decompositions, the top goal *manage internet shop*, and all the softgoals of the model. For all the other (leaf) goals we assume they have full evidence for satisfaction as initial assignment. For each experiment, the table reports the initial (Init) and final (Fin) values assumed by each goal.

In the first experiment (Exp1) we adopt the goal *DB querying* as the choice to achieve *manage item searching*, the goal *pick available item* to achieve *select items*, and the goal *classic communication handled* to achieve *get identification details*. The result is that the top goal *manage internet shop* is fully satisfied (Sat(...)=F) and all the softgoals are at least partial satisfied (Sat(...)=P), except the softgoal *easy to use* that results partially denied (Den(...)=P). Notice also that this initial assignment produces a conflict for the *integrity* softgoal (Sat(...)=P and Den(...)=P). In the second experiment (Exp2) we adopt the goal *standard form order* instead of the goal *classic communication handled*. This mainly produce the result of moving the conflict from the softgoal *integrity* to the softgoal *privacy*. In the third experiment (Exp3) we decide to *manage item searching* using the *catalogue consulting* goal. The effect of this new assignment is that softgoal *easy to use* is now partially satisfied, but we have conflicts for softgoals *integrity* and *privacy*. Finally, in the fourth experiment (Exp4) we adopt *secure form order* instead of the *standard form order* goal. This has the effect that now we do not have conflicts and all the softgoals are at least partially satisfied.

Goals	Exp 1				Exp 2				Exp 3				Exp 4			
	Init		Fin		Init		Fin		Init		Fin		Init		Fin	
	S	D	S	D	S	D	S	D	S	D	S	D	S	D	S	D
DB querying	F		F		F		F									
catalogue consulting									F		F		F		F	
pick available item	F		F		F		F		F		F		F		F	
pre-order non available item																
classic communication handled	F		F													
standard form order					F		F		F		F					
secure form order													F		F	
manage internet shop			F				F				F				F	
privacy			P				P	P			P	P			P	
availability			P				P				P				P	
integrity			P	P			P				P	P			P	
usability			P				P				P				P	
adaptability			F				F				F				F	
easy to use				P				P			P				P	
security			P				P				P				P	

Table 2.1: Evaluating alternatives in the goal model of Figure 2.4.

Table 2.1 reports just an example of analysis and it is limited to the simple model of Figure 2.4. Also in the model we have used only symmetric relationships and we have not distinguished between relations $+_S$ and $+_D$ or $-_S$ and $-_D$. In real-life case studies, the goal models to be analyzed are usually more complex. For instance, in [RGM04], we have presented a goal model with more than hundred goals for the Trentino Public Transportation System, in which non symmetric relationships have been used.

The analysis presented above concerns only a goal model and do not consider the effects of a particular assignment to the goals of other goal models. This kind of analysis is called *intra actor* analysis since it does not involve goal models of other actors. Differently, the *inter actor* analysis extends the boundary of the analysis to the goal models of the other actors. So for instance, we could analyze the effects of an assignment of Table 2.1 to the softgoals of the goal model shown in Figure 2.2, such as *happy customers* and *improve quality of services*.

2.3.2 Backword Reasoning

Backword reasoning focuses on the *backward search* of the possible input values leading to some desired final value, under desired constraints. We set the desired final values of the target goals, and we want to find possible initial assignments to the input goals which would cause the desired final values of the target goals by forward propagation. We may also add some desired constraints, and decide to avoid strong/medium/weak conflicts.

So for instance, in the goal model of Figure 2.4 we may be interested in finding an assignment without any conflict such that the top goal *manage internet shop* and the softgoal *security* are both fully satisfied.

In [RGM04] we have presented a solution to the backward reasoning reducing the problem to that of propositional satisfiability (SAT) [ZM02]. The boolean variables of the formula Φ to be satisfied are all the values $\mathsf{FS}(G)$, $\mathsf{PS}(G)$, $\mathsf{FD}(G)$, $\mathsf{PD}(G)$ for every goal $G \in \mathcal{G}$, and Φ is written in the form:

$$\Phi := \Phi_{graph} \wedge \Phi_{outval} \wedge \Phi_{backward} \, [\wedge \Phi_{optional} \,], \tag{2.18}$$

where Φ_{graph} encodes the goal graph and the axioms presented in Section 2.2, Φ_{outval} represents the desired final output values and $\Phi_{backward}$ encodes the backward reasoning. (See [RGM04] for details.) The optional formula $\Phi_{optional}$ allows the user to impose some constraints on the possible values of the goals and to force some desired value(s).

[RGM04] also presents a variant to the approach that allows us to assign a cost value to the satisfaction (or deniability) to the goals and hence find a solution with the minimum overall cost. Thus, for instance, in the goal model of our *Medi@* shop, we may be interested in finding an assignment with the minimal cost able to guarantee the full satisfaction of the top goal *manage internet shop* and the softgoal *security*. This approach is based on a variant of SAT, namely Minimum-Weight Propositional Satisfiability (MW-SAT) [Lib00]. (See again [RGM04] for the details.)

In Tropos the backward reasoning is used to analyze goal models and find the set of goals at the minimum costs that if achieved they can guarantee the achievement of the desired top goals and softgoals. In other words, we find among the alternatives of the goal model those with the minimal cost that allow us to obtain our desired goals.

Goals	Exp 1				Exp 2				Exp 3				Exp 4			
	Init		Fin		Init		Fin		Init		Fin		Init		Fin	
	S	D	S	D	S	D	S	D	S	D	S	D	S	D	S	D
DB querying (3)															F	
catalogue consulting (6)							F				F					
pick available item (2)							F				F				F	
pre-order non available item (7)																
classic communication handled (4)																
standard form order (6)											F					
secure form order (8)							F								F	
manage internet shop	F				F		F		F		F		F		F	
privacy	F				P		P		P		P	P	P		P	
availability	F				P		P		P		P				P	
integrity	F				P		P		P		P	P			P	
usability	F				P		P		P		P				P	
adaptability	F				P		F		P		F				F	
easy to use	F				P		P		P		P					P
security	F				P		P		P		P				P	

Table 2.2: Backward reasoning with the goal model of Figure 2.4.

Table 2.2 presents the results of the backward reasoning in four different situations with the goal model of the *Medi@* shop presented in Figure 2.4. The cost of each alternative goal is reported near its label (e.g., the cost of the *DB querying* is 3.). In the first experiment we try to find an assignment at the minimal cost that allows to obtain the full satisfaction of the top goal *manage internet shop* and all the softgoals. Unfortunately, no solution exists and this is due to the fact that almost all the softgoals receive only (+) and no (++) contributions. In the second experiment we require the full satisfaction of the top goal *manage internet shop* and partial satisfaction of all the softgoals. The solution at the minimum cost results the full satisfaction for *catalogue consulting*, *pick available item* and *secure form order*. In third experiments, we relaxed the constraint of avoiding conflicts and we obtain that now the solution includes the full satisfaction of the goal *standard form order* instead of the goal *secure form order*. Of course, now in the final values of the target goals we have conflicts, and in particular a conflict for the goal *privacy* (Sat(...)=P and Den(...)=P) and the goal *integrity* (Sat(...)=P and Den(...)=P). In the final experiment we imposed only the full satisfaction for the goal *manage internet shop* and the softgoal *privacy*. The solution with no conflicts is reported in table.

Also for backward reasoning the analysis can be extended to the goal models outside the boundary of the single actor. In this case the desired values can be assigned to (soft)goals of different goal models and the final solution will include goals of one or more goal models.

2.4 Goal Reasoning Tool

Forward and backward reasoning is supported in Tropos by the Goal Reasoning Tool (GR-Tool). Basically, the GR-Tool (Figure 2.6) is graphical tool in which it is possible to draw the goal models and run the algorithms and tools for forward and backward reasoning.

The algorithms for the forward reasoning, already presented in [GNMR02, GNMR03], have been fully developed in java and are embedded in the GR-Tool.

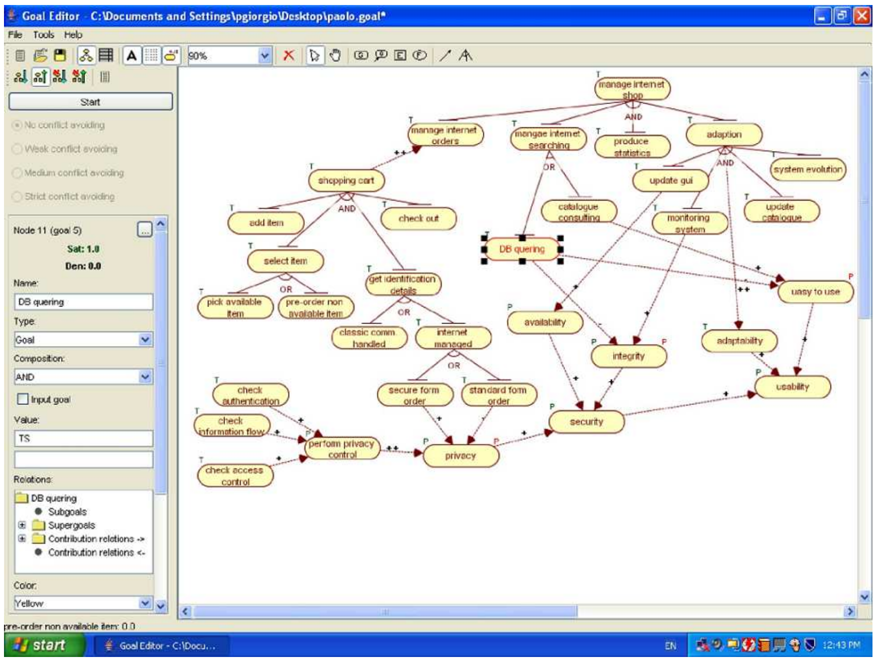


Figure 2.6: A snapshot of the GR-Tool

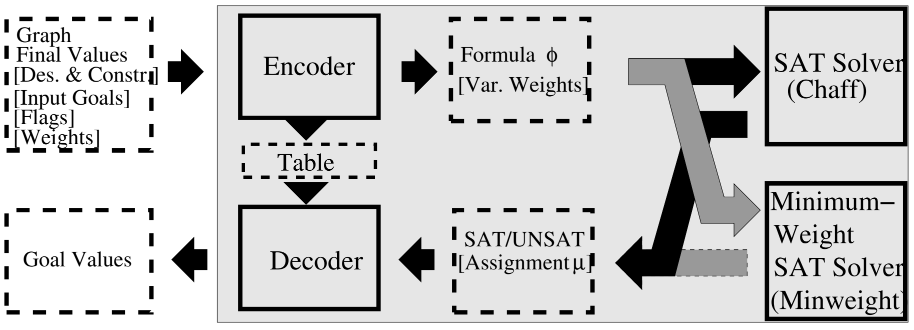


Figure 2.7: Schema of GOALSOLVE (black arrows) and GOALMINSOLVE (gray arrows).

For the backward reasoning we have implemented a tool called GOALSOLVE. The schema of GOALSOLVE is reported in Figure 2.7 (black arrows). GOALSOLVE takes as input a representation the goal graph, a list of desired final values and, optionally, a list of user desiderata and constraint and a list of goals which have to be considered as input. The user may also activate some flags for switching on the various levels of "avoiding conflicts".

The first component of GOALSOLVE is an encoder that generates the boolean CNF formula Φ as briefly described in the previous section, plus a correspondence table `Table` between goal values and their correspondent boolean variable. Φ is given as input to the SAT solver CHAFF [MMZ$^+$01], which returns either "UNSAT" if Φ is unsatisfiable, or "SAT" plus a satisfying assignment μ if Φ is satisfiable. Then a decoder uses `Table` to decode back the resulting assignment into the set of goal values.

In order to deal the minimum cost solutions, we have implemented a variant of GOALSOLVE, called GOALMINSOLVE, for the search of the goal values of *minimum cost*. The schema of GOALMINSOLVE is reported in Figure 2.7 (gray arrows). Unlike GOALSOLVE, GOALMINSOLVE takes as input also a list of integer weights $W(val(G))$ for the goal values, with some default options. The encoder here encodes also the input weight list into a list of weights for the corresponding boolean variables of Φ. Both Φ and the list of weights are given as input to the minimum-weight SAT solver MINWEIGHT [Lib00], which returns either "UNSAT" if Φ is unsatisfiable, or "SAT' plus a minimum-weight satisfying assignment μ if Φ is satisfiable. The decoder then works as in GOALSOLVE.

Notice that, in general, there may be many satisfying assignments —up to exponentially many— corresponding to solutions for the problem. In a typical session with GOALSOLVE or GOALMINSOLVE, the user may want to work first with the "avoiding conflicts" flags, starting from the most restrictive down to the least restrictive, until the problem admits solution. (E.g., it often the case that no solution avoiding all conflicts exists, but if one allows for weak and/or medium conflicts a solution exists.) Then, once the level of conflict avoidance is fixed, the user may want to work on refining the solution obtained, by iteratively adding positive and negative values in the list of desiderata and constraints, until a satisfactory solution is found.

Chapter 3

Semantic Matching for Goal Diagrams

3.1 Related Work

At present, there exists a line of semi-automated schema matching systems, see, for instance [DLD$^+$04, KN03, DR02, MBR01, MGMR02, BCV99, GATTM05]. A good survey and a classification of matching approaches up to 2001 is provided in [RB01], while an extension of its schema-based part and a user-centric classification of matching systems is provided in [SE05].

In particular, for individual matchers, [SE05] introduces the following criteria which allow for detailing further (with respect to [RB01]), the element and structure level of matching: *syntactic techniques* (these interpret their input as a function of their sole structures following some clearly stated algorithms, e.g., iterative fix point computation for matching graphs), *external techniques* (these exploit external resources of a domain and common knowledge, e.g., WordNet[Mil95]), and *semantic techniques* (these use formal semantics, e.g., model-theoretic semantics, in order to interpret the input and justify their results).

The distinction between the hybrid and composite matching algorithms of [RB01] is useful from an architectural perspective. [SE05] extends this work by taking into account how the systems can be distinguished in the matter of considering the mappings and the matching task, thus representing the end-user perspective. In this respect, the following criteria are proposed: *mappings as solutions* (these systems consider the matching problem as an optimization problem and the mapping is a solution to it, e.g., [MGMR02, EV04]); *mappings as theorems* (these systems rely on semantics and require the mapping to satisfy it, e.g., the approach proposed in this chapter); *mappings as likeness clues* (these systems produce only reasonable indications to a user for selecting the mappings, e.g., [MBR01, DR02]).

Let us consider some recent schema-based state of the art systems in light of the above criteria.

Rondo. The Similarity Flooding (SF) [MGMR02] approach, as implemented in Rondo [MRB03], utilizes a hybrid matching algorithm based on the ideas of similarity propagation. Schemas are presented as directed labeled graphs. The algorithm exploits only syntactic techniques at the element and structure level. It starts from the string-based comparison (common prefixes, suffixes tests) of the node's labels to obtain an initial mapping which is further refined within the fix-point computation. SF considers the mappings as a solution to a clearly stated optimization problem.

Cupid. Cupid [MBR01] implements a hybrid matching algorithm comprising syntactic techniques at the element (e.g., common prefixes, suffixes tests) and structure level (e.g., tree matching weighted by leaves). It also exploits external resources, in particular, a precompiled thesaurus. Cupid falls into the mappings as likeness clues category.

COMA. COMA [DR02] is a composite schema matching system which exploits syntactic and external techniques. It provides a library of matching algorithms; a framework for combining obtained results, and a platform for the evaluation of the effectiveness of the different matchers. The matching library is extensible, it contains 6 elementary matchers, 5 hybrid matchers, and one reuse-oriented matcher. Most of them implement string-based techniques (affix, n-gram, edit distance, etc.); others share techniques with Cupid (tree matching weighted by leaves, thesauri look-up, etc.); reuse-oriented is a completely novel matcher, which tries to reuse previously obtained results for entire new schemas or for its fragments. Distinct features of COMA with respect to Cupid, are a more flexible architecture and a possibility of performing iterations in the matching process. COMA falls into the mappings as likeness clues category.

3.2 Semantic Matching

We focus on tree-like structures, e.g., XML schemas. Real-world schemas are seldom trees, however, there are (optimized) techniques, transforming a graph representation of a schema into a tree representation, e.g., the graph-to-tree operator of Protoplasm [BMPQ04].

We call *concept of a label* the propositional formula which stands for the set of data instances that one would classify under a label it encodes. We call *concept at a node* the propositional formula which represents the set of data instances which one would classify under a node, given that it has a certain label and that it is in a certain position in a tree.

The semantic matching approach can discover the following semantic relations between the concepts of nodes of the two schemas: *equivalence* (=); *more general* ($\sqsupseteq$); *less general* ($\sqsubseteq$); *disjointness* ($\perp$). When none of the relations holds, the special *idk* (I don't know) relation is returned. The relations are ordered according to decreasing binding strength, i.e., from the strongest (=) to the weakest (*idk*), with more general and less general relations having equal binding power. The semantics of the above relations are the obvious set-theoretic semantics.

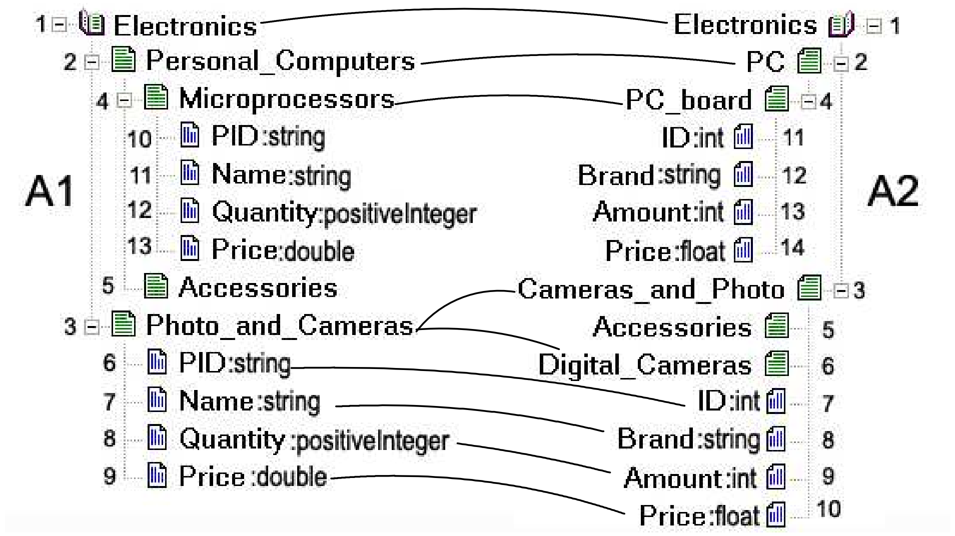


Figure 3.1: Two XML schemas and some of the mappings

A *mapping element* is a 4-tuple $\langle ID_{ij}, n1_i, n2_j, R\rangle$, i=1,...,N1; j=1,...,N2; where ID_{ij} is a unique identifier of the given mapping element; $n1_i$ is the i-th node of the first tree, N1 is the number of nodes in the first tree; $n2_j$ is the j-th node of the second tree, N2 is the number of nodes in the second tree; and R specifies a semantic relation which may hold between the *concepts of nodes* $n1_i$ and $n2_j$. *Semantic matching* can then be defined as the following problem: given two trees T1, T2 compute the N1 $\times$ N2 mapping elements $\langle ID_{i,j}, n1_i, n2_j, R'\rangle$, with $n1_i \in$ T1, i=1,...,N1, $n2_j \in$ T2, j=1,...,N2 and R' the strongest semantic relation holding between the concepts of nodes $n1_i, n2_j$.

3.2.1 The Tree Matching Algorithm

We summarize the algorithm for semantic schema matching via a running example. We consider the two simple XML schemas shown in Figure 3.1.

Let us introduce some notation (see also Figure 3.1). Numbers are the unique identifiers of nodes. We use "C" for concepts of labels and concepts at nodes. Also we use "$C1$" and "$C2$" to distinguish between concepts of labels and concepts at nodes in tree 1 and tree 2 respectively. Thus, in A1, $C1_{Photo\_and\_Cameras}$ and $C1_3$ are, respectively, the concept of the label $Photo\_and\_Cameras$ and the concept at node 3.

The algorithm takes as input two schemas and computes as output a set of mapping elements in four macro steps. The first two steps represent the pre-processing phase, while the third and the fourth steps are the element level and structure level matching respectively.

Step 1. For all labels *L* in the two trees, compute *concepts of labels*. We think of labels at nodes as concise descriptions of the data that is stored under the nodes. We compute the meaning of a label at a node by taking as input a label, by analyzing its real-world semantics, and by returning as output a concept of the label, C_L. Thus, for example, by writing $C_{Cameras\_and\_Photo}$ we move from the natural language ambiguous label $Cameras\_and\_Photo$ to the concept $C_{Cameras\_and\_Photo}$, which codifies explicitly its intended meaning, namely the data which is about cameras and photo.

Technically, we codify concepts of labels as propositional logical formulas. First, we chunk labels into *tokens*, e.g., $Photo\_and\_Cameras \rightarrow \langle photo, and, cameras \rangle$; and then, we extract *lemmas* from the tokens, e.g., $cameras \rightarrow camera$. Atomic formulas are WordNet [Mil95] *senses* of lemmas obtained from single words (e.g., cameras) or multiwords (e.g., digital cameras). Complex formulas are built by combining atomic formulas using the connectives of set theory. For example, $C2_{Cameras\_and\_Photo}$ = $\langle Cameras$,-$senses_{WN\#2} \rangle \sqcup \langle Photo, senses_{WN\#1} \rangle$, where $senses_{WN\#2}$ is taken to be disjunction of the two senses that WordNet attaches to *Cameras*, and similarly for *Photo*. Notice that the natural language conjunction "and" has been translated into the logical disjunction "$\sqcup$".

From now on, to simplify the presentation, we assume that the propositional formula encoding the concept of label is the label itself. We use numbers "1" and "2" as subscripts to distinguish between trees in which the given concept of label occurs. Thus, for example, $Cameras\_and\_Photo_2$ is a notational equivalent of $C2_{Cameras\_and\_Photo}$.

Step 2. For all nodes *N* in the two trees, compute *concepts of nodes*. In this step we analyze the meaning of the positions that the labels at nodes have in a tree. By doing this we *extend* concepts of labels to *concepts of nodes*, C_N. This is required to capture the knowledge residing in the structure of a tree, namely the context in which the given concept at label occurs. For example, in A2, when we write C_6 we mean the concept describing all the data instances of the electronic photography products which are digital cameras.

Technically, concepts of nodes are written in the same propositional logical language as concepts of labels. XML schemas are hierarchical structures where the path from the root to a node uniquely identifies that node (and also its meaning). Thus, following an *access criterion* semantics [Gua04], the logical formula for a concept at node is defined as a conjunction of concepts of labels located in the path from the given node to the root. For example, $C2_6 = Electronics_2 \sqcap Cameras\_and\_Photo_2 \sqcap Digital\_Cameras_2$.

Step 3. For all pairs of labels in the two trees, compute *relations* among *concepts of labels*. Relations between concepts of labels are computed with the help of a library of element level semantic matchers. These matchers take as input two atomic concepts of labels and produce as output a semantic relation between them. Some of the them are re-implementations of the well-known matchers used in Cupid and COMA. The most important difference is that our matchers return a semantic relation (e.g., =, $\sqsupseteq$, $\sqsubseteq$), rather an affinity level in the [0,1] range, although sometimes using customizable thresholds.

The element level semantic matchers are briefly summarized in Table 3.1. The first column contains the names of the matchers. The second column lists the order in which they are executed. The third column introduces the matcher's approximation level. The

Table 3.1: Element level semantic matchers.

Matcher name	**Execution order**	**Approximation level**	**Matcher type**	**Schema info**
Prefix	2	2	String-based	Labels
Suffix	3	2	String-based	Labels
Edit distance	4	2	String-based	Labels
Ngram	5	2	String-based	Labels
Text corpus	12	3	String-based	Labels + corpus
WordNet	1	1	Sense-based	WordNet senses
Hierarchy distance	6	3	Sense-based	WordNet senses
WordNet gloss	7	3	Gloss-based	WordNet senses
Extended WordNet gloss	8	3	Gloss-based	WordNet senses
Gloss comparison	9	3	Gloss-based	WordNet senses
Extended gloss comparison	10	3	Gloss-based	WordNet senses
Extended semantic gloss comparison	11	3	Gloss-based	WordNet senses

relations produced by a matcher with the first approximation level are always correct. For example, *name* $\sqsupseteq$ *brand* returned by *WordNet*. In fact, according to WordNet *name* is a hypernym (superordinate word) to *brand*. Notice that in WordNet *name* has 15 senses and *brand* has 9 senses. We use some sense filtering techniques to discard the irrelevant senses for the given context, see [MSS03] for details. The relations produced by a matcher with the second approximation level are likely to be correct (e.g., *net* = *network*, but *hot* = *hotel* by *Suffix*). The relations produced by a matcher with the third approximation level depend heavily on the context of the matching task (e.g., *cat* = *dog* by *Extended gloss comparison* in the sense that they are both *pets*). Notice that matchers are executed following the order of increasing approximation. The fourth column reports the matcher's type, while the fifth column describes the matcher's input.

As from Table 3.1, we have three main categories of matchers. *String-based* matchers have two labels as input (with exception of *Text corpus* which takes in input also a text corpus). These compute only equivalence relations (e.g., equivalence holds if the weighted *distance* between the input strings is lower than a threshold). *Sense-based* matchers have two WordNet senses in input. The *WordNet* matcher computes equivalence, more/less general, and disjointness relations; while *Hierarchy distance* computes only the equivalence relation. *Gloss-based* matchers also have two WordNet senses as input, however they exploit techniques based on comparison of textual definitions (*glosses*) of the words whose senses are taken in input. These compute, depending on a particular matcher, the equivalence, more/less general relations. The result of step 3 is a matrix of the relations holding between concepts of labels. A part of this matrix for the example of Figure 3.1 is shown in Table 3.2.

Step 4. For all pairs of nodes in the two trees, compute *relations* among *concepts of nodes*. During this step, we initially reformulate the tree matching problem into a set of node matching problems (one problem for each pair of nodes). Finally, we translate each node matching problem into a propositional validity problem. Let us discuss in detail the tree matching algorithm, see Algorithm 1 for the pseudo-code.

Table 3.2: The matrix of semantic relations holding between concepts of labels.

	$Cameras_2$	$Photo_2$	$Digital\_Cameras_2$
$Photo_1$	idk	$=$	idk
$Cameras_1$	$=$	idk	$\sqsupseteq$

Algorithm 1 The tree matching algorithm

```
1: Node: struct of
2:         int nodeId;
3:         String label;
4:         String cLabel;
5:         String cNode;
6: String[ ][ ] treeMatch(Tree of Nodes source, target)
7: Node sourceNode, targetNode;
8: String[ ][ ] cLabsMatrix, cNodesMatrix, relMatrix;
9: String axioms, context_1, context_2;
10: int i,j;
11: cLabsMatrix = fillCLabMatrix(source, target);
12: for each sourceNode ∈ source do
13:    i = getNodeId(sourceNode);
14:    context_1 = getCnodeFormula(sourceNode);
15:    for each targetNode ∈ target do
16:       j = getNodeId(targetNode);
17:       context_2 = getCnodeFormula(targetNode);
18:       relMatrix = extractRelMatrix(cLabMatrix, sourceNode, targetNode);
19:       axioms = mkAxioms(relMatrix);
20:       cNodesMatrix[i][j] = nodeMatch(axioms, context_1, context_2);
21:    end for
22: end for
23: return cNodesMatrix;
```

In line 6, the treeMatch function takes two trees of Nodes (source and target) in input. It starts from the element level matching. Thus, in line 11, the matrix of relations holding between concepts of labels (cLabsMatrix) is populated by the fillCLabsMatrix function which uses the library of element level matchers. We run two loops over all the nodes of source and target trees in lines 12-22 and 15-21 in order to formulate all our node matching problems. Then, for each node matching problem we take a pair of propositional formulas encoding concepts of nodes and relevant relations holding between concepts of labels using the getCnodeFormula and extractRelMatrix functions respectively. The former are memorized as context$_1$ and context$_2$ in lines 14 and 17. The latter are memorized in relMatrix in line 18. In order to reason about relations between concepts of nodes, we build the premises (axioms) in line 19. These are a conjunction of the concepts of labels which are related in relMatrix. For example, the task of matching $C1_3$ and $C2_6$, requires the following axioms: $(Electronics_1 = Electronics_2) \sqcap (Cameras_1 = Cameras_2) \sqcap (Photo_1 = Photo_2) \sqcap (Cameras_1 \sqsupseteq Digital\_Cameras_2)$. Finally, in line 20, the relations holding between the concepts of nodes are calculated by nodeMatch and are reported in line 23 as a bidimensional array (cNodesMatrix). A part of this matrix for the example of Figure 3.1 is shown in Table 3.3.

Table 3.3: The matrix of semantic relations holding between concepts of nodes (the matching result).

	$C2_1$	$C2_2$	$C2_3$	$C2_4$	$C2_5$	$C2_6$
$C1_3$	$\sqsubseteq$	*idk*	$=$	*idk*	$\sqsupseteq$	$\sqsupseteq$

3.2.2 The Node Matching Algorithm

We translate the node matching problem into a propositional validity problem. Semantic relations are translated into propositional connectives using the rules described in Table 3.4 (second column). The criterion for determining whether a relation holds between concepts of nodes is the fact that it is entailed by the premises. Thus, we have to prove that the following formula:

$$axioms \longrightarrow rel(context_1, context_2) \tag{3.1}$$

is valid, namely that it is *true* for all the truth assignments of all the propositional variables occurring in it. *axioms*, $context_1$, and $context_2$ are as defined in the tree matching algorithm. *rel* is the semantic relation that we want to prove holding between $context_1$ and $context_2$. The algorithm checks for the validity of formula (3.1) by proving that its negation, i.e., formula (3.2), is unsatisfiable.

$$axioms \wedge \neg rel(context_1, context_2) \tag{3.2}$$

Table 3.4 (third column) describes how formula (3.2) is translated before testing each semantic relation. Notice that (3.2) is in Conjunctive Normal Form (CNF), namely it is a conjunction of disjunctions of atomic formulas. In this case we assume that atomic formulas never occur negated, following what is common practice in building labels of, e.g., XML schemas. Also, notice that $a = b$ iff both $a \sqsubseteq b$ and $b \sqsubseteq a$ hold, therefore we do not need to test the equivalence relation separately.

Table 3.4: The relationship between semantic relations and propositional formulas.

$rel(a,b)$	Translation of $rel(a,b)$ into propositional logic	Translation of formula (3.2) into Conjunctive Normal Form
$a = b$	$a \leftrightarrow b$	N/A
$a \sqsubseteq b$	$a \rightarrow b$	$axioms \wedge context_1 \wedge \neg context_2$
$a \sqsupseteq b$	$b \rightarrow a$	$axioms \wedge context_2 \wedge \neg context_1$
$a \bot b$	$\neg(a \wedge b)$	$axioms \wedge context_1 \wedge context_2$

The pseudo-code of a basic solution for the node matching algorithm is provided in Algorithm 2. Let us analyze it in detail. In lines 110 and 140, the nodeMatch function constructs the formulas for testing the less general and more general relations. In lines 120 and 150, it converts them into CNF, while in lines 130 and 160, it checks formulas in

Algorithm 2 The node matching algorithm

```
100. String nodeMatch(String axioms, context_1, context_2)
110. String formula = And(axioms, context_1, Not(context_2));
120. String formulaInCNF = convertToCNF(formula);
130. boolean isLG = isUnsatisfiable(formulaInCNF);
140. formula = And(axioms, Not(context_1), context_2);
150. formulaInCNF = convertToCNF(formula);
160. boolean isMG = isUnsatisfiable(formulaInCNF);
170. if(isMG && isLG) then
180.     return "$=$";
190. endif
200. if (isLG) then
210.     return "$\sqsubseteq$";
220. endif
230. if (isMG) then
240.     return "$\sqsupseteq$";
250. endif
260. formula = And(axioms, context_1, context_2);
270. formulaInCNF = convertToCNF(formula);
280. boolean isOpposite = isUnsatisfiable(formulaInCNF);
290. if (isOpposite) then
300.     return "$\perp$";
310. else
320.     return "$idk$";
330. endif
```

CNF for unsatisfiability. If both relations hold, then the equivalence relation is returned (line 180). Finally, the same procedure is repeated for the disjointness relation. If all the tests fail, the idk relation is returned (line 320).

In order to check the unsatisfiability of a propositional formula in a basic version of our NodeMatch algorithm we use the standard DPLL-based SAT solver [Ber, DLL62]. From the example in Figure 1, trying to prove that $C2_6$ is less general than $C1_3$, requires constructing formula (3.3), which turns out to be unsatisfiable, and therefore, the less generality holds.

$$\begin{aligned}
&((Electronics_1 \leftrightarrow Electronics_2) \wedge (Photo_1 \leftrightarrow Photo_2) \wedge \\
&(Cameras_1 \leftrightarrow Cameras_2) \wedge (Digital\_Cameras_2 \rightarrow Cameras_1)) \wedge \\
&(Electronics_2 \wedge (Cameras_2 \vee Photo_2) \wedge Digital\_Cameras_2) \wedge \neg \\
&(Electronics_1 \wedge (Photo_1 \vee Cameras_1))
\end{aligned} \tag{3.3}$$

3.3 Efficient Semantic Matching

In this section we present a set of optimizations for the node matching algorithm. In particular, we show, that when dealing with *conjunctive concepts at nodes*, i.e., the concept of node is a conjunction (e.g., $C1_2$ = $Electronics_1 \wedge Personal\_Computers_1$), these node matching tasks can be solved in linear time. When we have *disjunctive concepts at nodes*, i.e., the concept of node contains both conjunctions and disjunctions in any order (e.g., $C2_6 = Electronics_2 \wedge (Cameras_2 \vee Photos_2) \wedge Digital\_Cameras_2$), we use techniques avoiding the exponential space explosion which arises due to the conversion of disjunctive formulas into CNF. This modification is required since all state of the art SAT deciders take CNF formulas in input.

3.3.1 Conjunctive concepts at nodes

Let us make some observations with respect to Table 3.4. The first observation is that *axioms* remains the same for all the tests, and it contains only clauses with two variables. In the worst case, it contains $2 \cdot n_1 \cdot n_2$ clauses, where n_1 and n_2 are the number of atomic concepts of labels occurred in $context_1$ and $context_2$ respectively. The second observation is that the formulas for less and more generality tests are very similar and they differ only in the negated context formula (e.g., in the less generality test $context_2$ is negated). This means that formula (3.1) contains one clause with n_2 variables plus n_1 clauses with one variable. In the case of disjointness test $context_1$ and $context_2$ are not negated. Therefore, formula (3.1) contains $n_1 + n_2$ clauses with one variable. For lack of space, let us only consider tests for more/less general relations.

Less and more generality tests.

Using the above observations, formula (3.1), with respect to the less/more generality tests, can be represented as follows:

$$\overbrace{\bigwedge_{0}^{n*m} (\neg A_s \vee B_t) \wedge \bigwedge_{0}^{n*m} (A_k \vee \neg B_l) \wedge \bigwedge_{0}^{n*m} (\neg A_p \vee \neg B_r)}^{axioms} \wedge \overbrace{\bigwedge_{i=1}^{n} A_i}^{context_1} \wedge \overbrace{\bigvee_{j=1}^{m} \neg B_j}^{\neg context_2} \tag{3.4}$$

where n is the number of variables in $context_1$, m is the number of variables in $context_2$. The Ai's belong to $context_1$, and the Bj's belong to $context_2$. s, k, p are in the [0..n] range, while t, l, r are in the [0..m] range. Axioms can be empty. Formula (3.4) is composed of clauses with one or two variables plus one clause with possibly more variables (the clause corresponding to the negated context). Notice that formula (3.4) is *Horn*, i.e., each clause contains at most one positive literal. Therefore, its satisfiability can be decided in linear time by the *unit resolution rule*. DPLL-based SAT solvers in this case require quadratic time. In order to understand how the linear time algorithm works, let us suppose that we want to check if $C1_4$ is less general than $C2_4$. Formula (3.4) in this case is as follows:

$$\begin{array}{l}
((\neg \boldsymbol{Electronics}_1 \vee Electronics_2) \wedge (\boldsymbol{Electronics}_1 \vee \neg Electronics_2) \wedge \\
(\neg \boldsymbol{Personal\_Computers}_1 \vee PC_2) \wedge (\boldsymbol{Personal\_Computers}_1 \vee \neg PC_2) \wedge \\
(\neg \boldsymbol{Microprocessors}_1 \vee \neg PC\_board_2)) \wedge \\
\boldsymbol{Electronics}_1 \wedge \boldsymbol{Personal\_Computers}_1 \wedge \boldsymbol{Microprocessors}_1 \wedge \\
(\neg Electronics_2 \vee \neg PC_2 \vee \neg PC\_board_2)
\end{array} \tag{3.5}$$

where the variables from $context_1$ are written in bold. First, we assign true to all the unit clauses occurring in (3.5) positively. Notice that these are all and only the clauses in $context_1$, namely, ***Electronics***$_1$, ***Personal_Computers***$_1$, and ***Microprocessors***$_1$. This allows us to discard the clauses where variables from $context_1$ occur positively, namely, (***Electronics***$_1$ $\vee\neg Electronics_2$) and (***Personal_Computers***$_1$ $\vee\neg PC_2$). Thus, the resulting formula is as follows:

$$\begin{gathered}(Electronics_2 \wedge PC_2 \wedge \neg PC\_board_2) \wedge \\ (\neg Electronics_2 \vee \neg PC_2 \vee \neg PC\_board_2)\end{gathered} \tag{3.6}$$

Formula (3.6) does not contain any variable from $context_1$. By assigning true to *Electronics*$_2$ and false to *PC_board*$_2$ we do not determine a contradiction, and therefore, (3.6) is satisfiable.

For formula (3.6) to be unsatisfiable, all the variables occurring in the negation of $context_2$, namely, $(\neg Electronics_2 \vee \neg PC_2 \vee \neg PC\_board_2)$ should occur positively in the unit clauses obtained after resolving *axioms* with the unit clauses in $context_1$, namely, $Electronics_2$ and PC_2. For this to happen, for any Bj there must be a clause of the form $\neg Ai \vee Bj$ in *axioms*. Formulas of the form $\neg Ai \vee Bj$ occur in (3.4) iff we have the axioms of type $Ai = Bj$ and $Ai \sqsubseteq Bj$. These considerations suggest the following algorithm for testing satisfiability:

- *Step 1.* Create an array of size m. Each entry in the array stands for one Bj in (3.4).
- *Step 2.* For each axiom of type $Ai = Bj$ and $Ai \sqsubseteq Bj$ mark the corresponding Bj.
- *Step 3.* If all the Bj's are marked, then the formula is unsatisfiable.

Thus, nodeMatch can be optimized by using Algorithm 3. The numbers on the left indicate where the new code must be positioned in Algorithm 2. fastHornUnsatCheck implements the three steps above. Step 1 is performed in lines 402 and 403. In lines 404-409, a loop on axioms implements Step 2. The final loop in lines 410-416 implements Step 3.

3.3.2 Disjunctive concepts at nodes

Now, we allow for the concepts of nodes to contain conjunctions and disjunctions in any order. As from Table 3.4, *axioms* is the same for all the tests. However, $context_1$ and $context_2$ may contain any number of disjunctions. Some of them are coming from the concepts of labels, while others may appear from the negated $context_1$ or $context_2$ (e.g., see less/more generality tests). With disjunctive concepts at nodes, formula (3.1) is a full propositional formula, and hence, no hypothesis can be made on its structure. Thus, its satisfiability must be tested by using a standard SAT decider.

Algorithm 3 Optimizations: less/more generality tests

```
101. if (context$_1$ and context$_2$ are conjunctive) then
102.    isLG = fastHornUnsatCheck(context$_1$, axioms, "$\sqsubseteq$");
103.    isMG = fastHornUnsatCheck(context$_2$, axioms, "$\sqsupseteq$");
104. endif

401. boolean fastHornUnsatCheck(String context, axioms, rel)
402. int m = getNumOfVar(String context);
403. boolean array[m];
404. for each axiom $\in$ axioms do
405.    if (getAType(axiom) = {"=" || rel}) then
406.       int j = getNumberOfSecondVariable(axiom);
407.       array[j] = true;
408.    endif
409. endfor
410. for (i=0; i<m; i++) do
411.    if (!array[i]) then
412.       return false;
413.    else
414.       return true;
415.    endif
416. endfor
```

In order to avoid the exponential space explosion, which may arise when converting formula (3.1) into CNF, we apply a set of structure preserving transformations [PG86]. The main idea is to replace disjunctions occurring in the original formula with newly introduced variables and to explicitly state that these variables imply the subformulas they substitute. Therefore, the size of the propositional formula in CNF grows linearly with respect to the number of disjunctions in the original formula. Thus, nodeMatch should be optimized by replacing all the calls to convertToCNF with calls to optimizedConvertToCNF.

3.4 Semantic Matching with Attributes

XML elements may have attributes. Attributes are $\langle attribute - name, type \rangle$ pairs associated with elements. Names for the attributes are usually chosen such that they describe the roles played by the domains in order to ease distinguishing between their different uses. For example, in A1, the attributes PID and $Name$ are defined on the same domain $string$, but their intended uses are the internal (unique) product identification and representation of the official product's names respectively. There are no strict rules telling us when data should be represented as elements, or as attributes, and obviously there is always more than one way to encode the same data. For example, in A1, $PIDs$ are encoded as $string$s, while in A2, IDs are encoded as ints. However, both attributes serve for the same purpose of the unique product's identification. These observations suggest two possible ways to perform semantic matching with attributes: (i) taking into account datatypes, and (ii) ignoring datatypes.

The semantic matching approach is based on the idea of matching concepts, not their direct physical implementations, such as elements or attributes. If names of attributes and elements are abstract entities, therefore, they allow for building arbitrary concepts out of them. Instead, datatypes, being concrete entities, are limited in this sense. Thus, a plausible way to match attributes using the semantic matching approach is to discard the information about datatypes. In order to support this claim, let us consider both cases in turn.

3.4.1 Exploiting datatypes

In order to reason with datatypes we have created a *datatype ontology*, $\boldsymbol{O_D}$, specified in OWL [SWM04]. It describes the most often used XML schema built-in datatypes and relations between them. The backbone taxonomy of $\boldsymbol{O_D}$ is based on the following rule: *the is-a relationship holds between two datatypes iff their value spaces are related by set inclusion.* Some examples of axioms of $\boldsymbol{O_D}$ are: $\mathsf{float} \sqsubseteq \mathsf{double}$, $\mathsf{int} \perp \mathsf{string}$, $\mathsf{anyURI} \sqsubseteq \mathsf{string}$, and so on. Let us discuss how datatypes are plugged within the four macro steps of the algorithm.

Steps 1,2. Compute concepts of labels and nodes. In order to handle attributes, we extend propositional logics with the quantification construct and datatypes. Thus, we compute concepts of labels and concepts of nodes as formulas in description logics (DL), in particular, using $\mathcal{ALC}(\mathcal{D})$ [Pan04]. For example, $C1_7$, namely, the concept of node describing all the string data instances which are the names of electronic photography products is encoded as $Electronics_1 \sqcap (Photo_1 \sqcup Cameras_1) \sqcap \exists Name_1.string$.

Step 3. Compute relations among concepts of labels. In this step we extend our library of element level matchers by adding a *Datatype* matcher. It takes as input two datatypes, it queries $\boldsymbol{O_D}$ and retrieves a semantic relation between them. For example, from axioms of $\boldsymbol{O_D}$, the *Datatype* matcher can learn that $\mathsf{float} \sqsubseteq \mathsf{double}$, and so on.

Step 4. Compute relations among concepts of nodes. In the case of attributes, the node matching problem is translated into a DL formula, which is further checked for its unsatisfiability using sound and complete procedures. Notice that in this case we have to test for modal satisfiability, not propositional satisfiability. The system we use is Racer [HMW]. From the example in Figure 1, trying to prove that $C2_{10}$ is less general than $C1_9$, requires constructing the following formula:

$$\begin{array}{l}((Electronics_1{=}Electronics_2)\sqcap(Photo_1{=}Photo_2)\sqcap\\ (Cameras_1{=}Cameras_2)\sqcap(Price_1{=}Price_2)\sqcap(float\sqsubseteq double))\sqcap\\ (Electronics_2\sqcap(Cameras_2\sqcup Photo_2)\sqcap\exists Price_2.float)\sqcap\neg\\ (Electronics_1\sqcap(Photo_1\sqcup Cameras_1)\sqcap\exists Price_1.double)\end{array} \tag{3.7}$$

It turns out that formula (3.7) is unsatisfiable. Therefore, $C2_{10}$ is less general than $C1_9$. However, this result is not what the user expects. In fact, both $C1_9$ and $C2_{10}$ describe prices of electronic products, which are photo cameras. The storage format of *prices* in A1 and A2 (i.e., *double* and *float* respectively) is not an issue at this level of detail.

Thus, another semantic solution of taking into account datatypes would be to build abstractions out of the datatypes, e.g., float, double, decimal should be abstracted to type numeric, while token, name, normalizedString should be abstracted to type string, and so on. However, even such abstractions do not improve the situation, since we may have, for example, an *ID* of type numeric in the first schema, and a conceptually equivalent *ID*, but of type string, in the second schema. If we continue building such abstractions, we result in having that numeric is equivalent to string in the sense that they are both datatypes.

The last observation suggests that for the semantic matching approach to be correct, we should assume, that all the datatypes are equivalent between each other. Technically, in order to implement this assumption, we should add corresponding axioms (e.g., float = double) to the premises of formula (3.1). On the one hand, with respect to the case of not considering datatypes (see, Section 5.2), such axioms do not affect the matching result from the quality viewpoint. On the other hand, datatypes make the matching problem computationally more expensive by requiring to handle the quantification construct.

3.4.2 Ignoring datatypes

In this case, information about datatypes is discarded. For example, $\langle Name, string\rangle$ becomes $Name$. Then, the semantic matching algorithm builds concepts of labels out of attribute's names in the same way as it does in the case of element's names, and so on. Finally, it computes mappings using the optimized algorithm of Section 4. A part of the cNodesMatrix with relations holding between attributes for the example of Figure 3.1 is presented in Table 3.5. Notice that this solution allows us for a mapping's computation not only between the attributes, but also between attributes and elements.

Table 3.5: Attributes: the matrix of semantic relations holding between concepts of nodes (the matching result).

	$C2_7$	$C2_8$	$C2_9$	$C2_{10}$
$C1_6$	$=$	*idk*	*idk*	*idk*
$C1_7$	*idk*	$\sqsupseteq$	*idk*	*idk*
$C1_8$	*idk*	*idk*	$=$	*idk*
$C1_9$	*idk*	*idk*	*idk*	$=$

The task of determining mappings typically represents a first step towards the ultimate goal of, for example, data translation, query mediation, data integration, agent communication, and so on. Although information about datatypes will be necessary for accomplishing an ultimate goal, we do not discuss this issue any further since in this chapter we concentrate only on the mappings discovery task.

3.5 Comparative Evaluation

In this section, we present the quality and performance evaluation of the matching system we have implemented, called S-Match. In particular, we validate basic and optimized versions of our system, called (S-Match$_b$) and (S-Match$_o$) respectively, and evaluate them against three state of the art matchers, namely Cupid [MBR01], COMA [DR02][1], and SF [MGMR02] as implemented in Rondo [MRB03]. All the systems under consideration are fairly comparable because they are all schema-based. They differ in the specific matching techniques they use and in how they compute mappings.

In our evaluation we have used five pairs of schemas: two artificial examples, a pair of product schemas (our running example, i.e., A1 vs. A2), a pair of purchase order schemas (CIDX vs. Excel), and a pair of parts of web directories (Google vs. Looksmart). Table 3.6 provides some indicators of the complexity of the test cases[2]. As match quality measures we have used the following indicators: *precision*, *recall*, *overall*, *F-measure* (see, [DR02]). *Precision* varies in the [0,1] range; the higher the value, the smaller is the set of wrong mappings (false positives) which have been computed. *Precision* is a correctness measure. *Recall* varies in the [0,1] range; the higher the value, the smaller is the set of correct mappings (true positives) which have not been found. *Recall* is a completeness measure. *F-measure* varies in the [0,1] range. The version computed here is the harmonic mean of *precision* and *recall*. It is a global measure of the matching quality, growing with it. *Overall* is an estimate of the post-match efforts needed for adding false negatives and removing false positives. *Overall* varies in the [-1,1] range; the higher it is, the less post-match efforts are needed. As a performance measure we have used *time*. It estimates how fast systems are when producing mappings fully automatically.

To provide a ground for evaluating the quality of match results, initially, the schemas have been manually matched to produce expert mappings. Then, the results computed by systems have been compared with expert mappings. There are three further observations that ensure a fair comparative study. The first observation is that Cupid, COMA, and Rondo can discover only the mappings which express similarity between schema elements. Instead, S-Match, among the others, discovers the disjointness relation which can be interpreted as strong dissimilarity in terms of the other systems under consideration. Therefore, we did not take into account the disjointness relations (e.g., $\langle ID_{4,4}, C1_4, C2_4, \perp \rangle$) when specifying the expert mappings. The second observation is that, since S-Match returns a matrix of relations, while all the other systems return a list of the best mappings, we used some filtering rules. More precisely we have the following two rules: (i) discard all the mappings where the relation is idk; (ii) return always the *core* relations, and discard relations whose existence is implied by the core relations. For example,

[1]We thank Phil Bernstein, Hong Hai Do, and Erhard Rahm for providing us with Cupid and COMA. In the evaluation we use the version of COMA described in [DR02]. A newer version of the system COMA++ exists but we do not have it.

[2]Source files, description of the test cases, and expert mappings can be found at http://www.dit.unitn.it/∼accord/, experiments section.

Table 3.6: Some indicators of the complexity of the test cases

	#nodes	max depth	#labels per tree	concepts of nodes
Artificial Example #1	250/500	16/15	250/500	conjunctive
Artificial Example #2	10/10	10/10	30/30	disjunctive
A1 vs. A2	13/14	4/4	14/15	conjunctive & disjunctive
CIDX vs. Excel	34/39	3/3	56/58	conjunctive & disjunctive
Google vs. Looksmart	706/1081	11/16	1048/1715	conjunctive & disjunctive

$\langle ID_{3,3}, C1_3, C2_3, =\rangle$ should be returned, while $\langle ID_{3,5}, C1_3, C2_5, \sqsupseteq\rangle$ should be discarded. Finally, whether S-Match returns the equivalence or subsumption relations does not affect the quality indicators. What only matters is the presence of the mappings standing for those relations.

In our experiments each test has two degrees of freedom: *directionality* and *use of oracles*. By directionality we mean here the direction in which mappings have been computed: from the first schema to the second one (forward direction), or vice versa (backward direction). For lack of space we report only the best results obtained with respect to directionality, and use of oracles allowed. We were not able to plug a thesaurus in Rondo, since the version we have is standalone, and it does not support the use of external thesauri. Thesauri of S-Match, Cupid, and COMA were expanded with terms necessary for a fair competition (e.g., expanding *uom* into *unitOfMeasure*, a complete list is available at the URL in footnote 2).

All the tests have been performed on a P4-1700, 512 MB of RAM, Windows XP, with no applications running but a single matching system. Notice, that the systems were limited to allocate no more than 512 MB of main memory. Also, all the tuning parameters (e.g., thresholds, strategies) of the systems were taken by default (e.g., for COMA we used NamePath and Leaves matchers combined in the Average strategy) for all the tests.

3.5.1 Test Cases

Let us discuss artificially designed problems in order to evaluate the performance of S-Match$_o$ in *ideal* conditions, namely when we have only conjunctive or disjunctive concepts of nodes. Since examples are artificial and our optimizations address only efficiency, not quality, we analyze here only the performance *time* of the systems, see, Figure 3.2 (Artificial Examples).

On the example with conjunctive concepts at nodes (Artificial Example #1), COMA performs 4 times slower and 15 times slower than S-Match$_b$ and S-Match$_o$ respectively. S-Match$_o$ runs around 29% faster than Rondo. Instead, Cupid runs out of memory.

On the example with disjunctive concepts at nodes (Artificial Example #2), S-Match$_o$ works around 4 orders of magnitude faster than S-Match$_b$, around 5 times faster than COMA, 1.6 times faster than Cupid, and as fast as Rondo. The significant improvement of

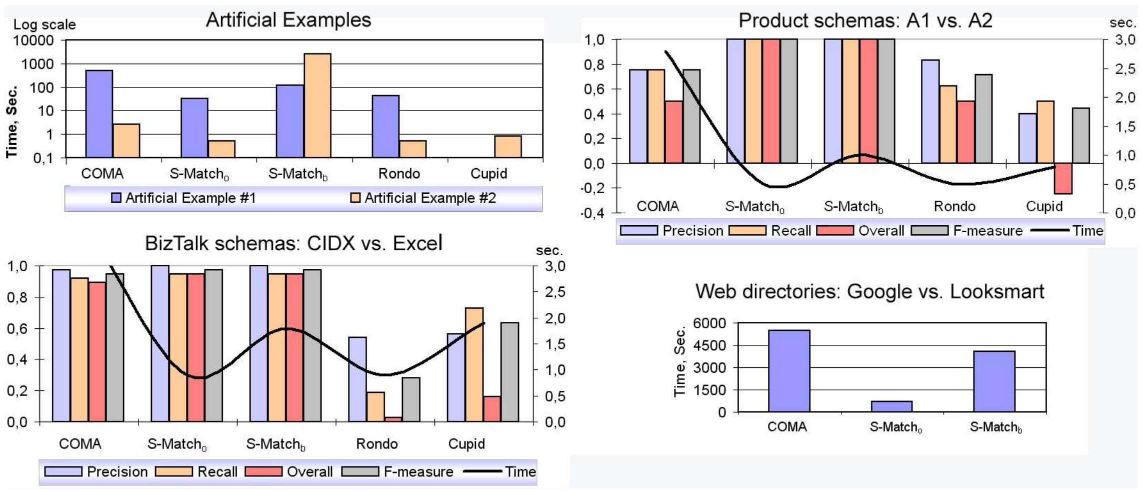


Figure 3.2: Evaluation Results

our optimized algorithm can be explained by considering that S-Match$_b$ does not control the exponential space explosion on such matching problems. In fact, the biggest formula in this case consists of about 118000 clauses. The optimization introduced in the Section 4.2 reduces this number to approximately 20-30 clauses.

We have then considered 3 matching problems, also involving real-world examples. Let us first discuss matching results from our running example, see, Figure 3.2 (Product schemas: A1 vs. A2). There, S-Match outperforms the other systems in terms of quality indicators. Since all the labels at nodes in the given test case were correctly encoded into propositional formulas, all the quality measures of S-Match reach their highest values. In fact, as discussed before, the propositional SAT solver is correct and complete. This means that once the element level matchers have found all and only the mappings, S-Match will return all of them and only the correct ones. Also, S-Match$_o$ works more than 5 times faster than COMA, 1.5 times faster than Cupid, and as fast as Rondo.

For a pair of BizTalk schemas: CIDX vs. Excel, S-Match performs as good as COMA and outperforms the other systems in terms of quality indicators. Also, S-Match$_o$ works more than 4 times faster than COMA, more than 2 times faster than Cupid, and as fast as Rondo.

For the biggest matching problem (Web Directories: Google vs. Looksmart), which contains hundreds and thousands of nodes, unfortunately, we did not have enough human resources to create expert mappings for this test case (we are still working on establishing them), and thus, for the moment we have evaluated only the performance *time*. S-Match$_o$ performs about 9 times faster than COMA, and about 7 times faster than S-Match$_b$. Rondo and Cupid run out of memory, therefore we do not report any results for them.

3.5.2 Evaluation Summary

Quality measures. Since most matching systems return similarity coefficients, rather than semantic relations, our qualitative analysis was based on the measures developed for those systems. Therefore, we had to omit information about the type of relations S-Match returns, and focus only on the number of present/ absent mappings. We totally discarded from our considerations the disjointness relation, however, its value should not be underestimated, because this relation reduces the search space. For the example of Figure 3.1, if Cupid would support the analysis of dissimilarity between schema elements, it could possibly recognize that $C1_4$ is disjoint (dissimilar) with $C2_4$, and then avoid false positives such as determining that $C1_{10}$ is similar to $C2_{11}$, and so on.

Pre-match efforts. Typically, these efforts include creating a precompiled thesaurus with relations among common and domain specific terms. On the one side, such a thesaurus can be further reused, since many schemas to be matched are similar to already matched schemas, especially if they are describing the same application domain. On the other side, for the first schemas to be matched from a novel domain, creation of such a thesaurus requires time. With this respect, exploiting an external resource of common and domain knowledge (e.g., WordNet) can significantly reduce the pre-match efforts. In the example of Figure 3.1, in order for Cupid to determine $C1_7$ as an appropriate match for $C2_8$, we have to add an entry <Hyp key="brand:name"> 0.7</Hyp> to its thesaurus, while S-Match obtains the knowledge of the hyponymy relation in the above case automatically from WordNet.

Performance measures. *Time* is a very important indicator, because when matching industrial-size schemas (e.g., with hundreds and thousands of nodes, which is quite typical for e-business applications), it shows scalability properties of the matchers and their potential to become an industrial-strength systems. It is also important in web applications, where some weak form of real time performance is required (to avoid having a user waiting too long for the system respond).

Chapter 4

ST-Tool: A CASE Tool for Security Requirements Engineering

4.1 Background

Secure Tropos [GMMZ04] is an agent-oriented software development methodology, tailored to describe both the organization and the system with respect to functional and security requirements. Secure Tropos extends the Tropos methodology [BGG^{+}04b] and has the concepts of actor, service (i.e. goal, task, resource) and social relationships for defining the obligations among actors. A description of these concepts is provided in [GMMZ04].

Various activities contribute to the acquisition of a first requirement model:

Actor modeling, which consists of identifying and analyzing both environment and system's actors.

Trust modeling, which consists of identifying actors which trust other actors for services, and actors which own services.

Delegation modeling, which consists of identifying actors which delegate to other actors the permission and/or execution on services.

Once the stakeholders have been identified, along with their goals and social relations, the analysis proceeds in order to enrich the model with further details. Goal refinement rests on the analysis of actor goals and is conducted by using AND/OR decomposition.

Due to lack of space, we have focused on the key modeling aspects of the framework and refer to [GMMZ04] for the introduction of the formal framework based on Datalog.

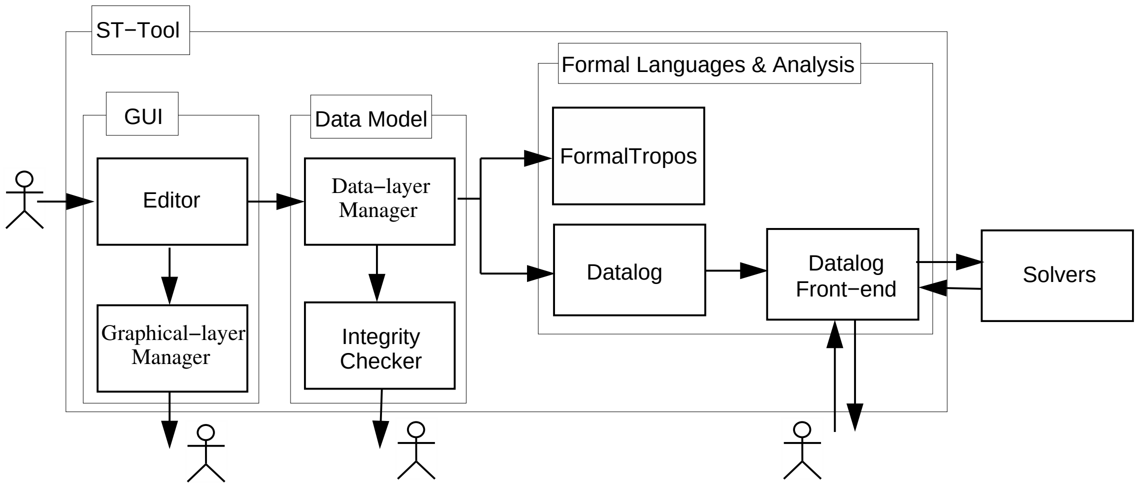


Figure 4.1: The Architecture Overview

4.2 Overview of ST-Tool

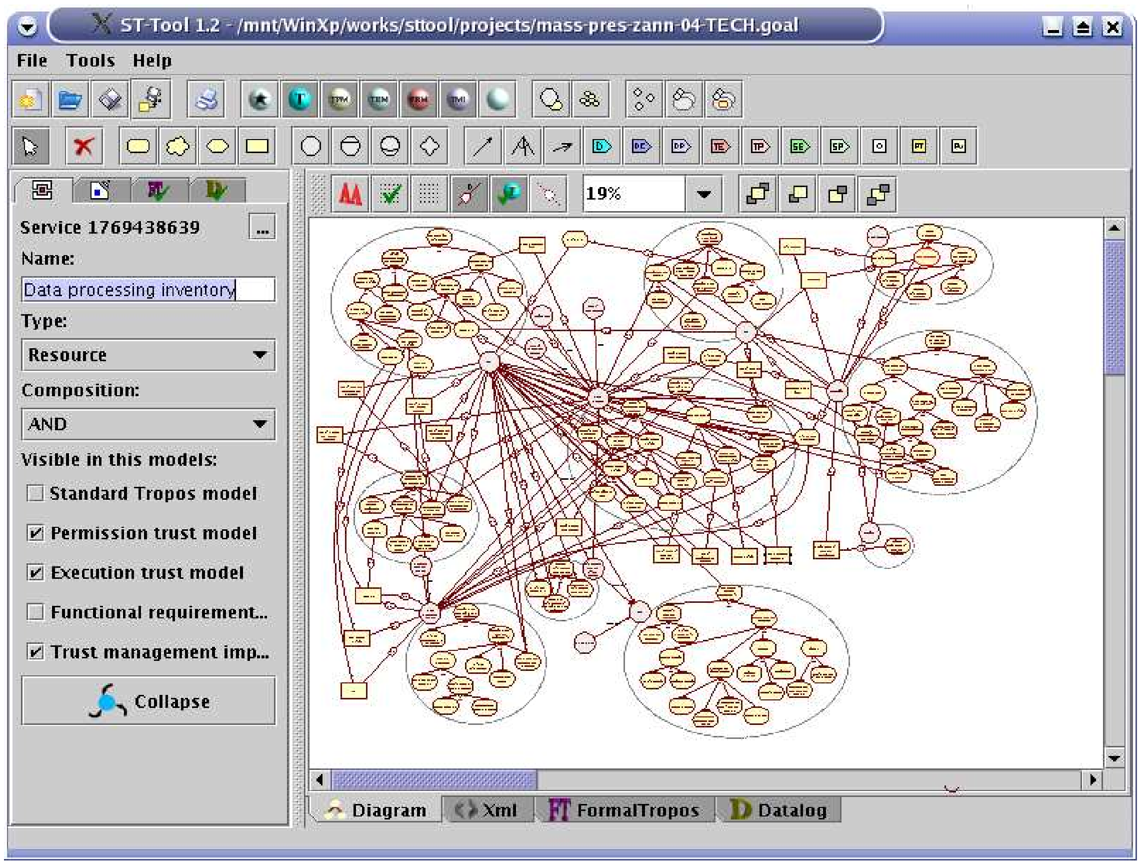
ST-Tool is mainly composed of two parts: ST-Tool kernel and external solvers. ST-Tool kernel has an architecture comprised of three major parts, each of which is comprised of modules. Next, we discuss these modules and their interconnections. In Fig. 4.1, the modules of ST-Tool are shown, their interrelations are also indicated.

ST-Tool provides a graphical user interface (GUI), through which all its components are managed. A screen shot is shown in Fig. 4.2. The GUI's key component is the *Editor Module*. This allows designers to edit Secure Tropos models as graphs where nodes are actors and services, and arcs are relationships. A second component is the *Graphical-layer Manager (GM) Module* that aims to manage graphical objects. It supports goal refinement by associating a goal diagram with each actor. Further, GM permits to display one or more views of a diagram at the same time (dependency model, delegation model, trust model).

The *Data-layer Manager (DM) Module* is responsible for maintaining data corresponding to graphical objects. Its main task is to manage misalignments between relationships and their graphical representation. A support for detecting errors and warnings during the modeling phase is provided by the *Integrity Checker Module*. Integrity Checker reports errors such as "orphan relations" (i.e. incomplete relations) and "isolated nodes" (i.e. services not involved in any relations). Warnings are different from errors since designers may be perfectly happy with a design that does not satisfy them. Integrity Checker reports warnings when more than one service have the same name.[1]

After drawing so many nice diagrams, designers may want to check whether the model satisfies some general desirable properties. The tool allows an automatic transformation

[1]More than one service with the same name are needed to model delegation and trust chains.


ST-Tool 1.2 - /mnt/WinXp/works/sttool/projects/mass-pres-zann-04-TECH.goal
File
Tools
Help
19%
Collapse
Diagram
Xml
FormalTropos
Datalog

Service 1769438639

Name: Data processing inventory

Type: Resource

Composition: AND

Visible in this models:

- [ ] Standard Tropos model
- [✓] Permission trust model
- [✓] Execution trust model
- [ ] Functional requirement...
- [✓] Trust management imp...

Figure 4.2: ST-Tool

from Secure Tropos graphical models into Datalog and Formal Tropos [FLP+03] specifications. These are performed by two different modules: the *Formal Tropos Module* and the *Datalog Module*. The resulting specifications are displayed by selecting the corresponding panel. Since the formal semantics of Secure Tropos is based on Datalog, we mainly focus on Datalog Module. The intuitive descriptions of systems are often incomplete, and need to be completed for a correct analysis [GMMZ04]. The *Datalog Front-end (DF) Module* provides support for model completing and checking by using external Datalog solvers. Essentially, DF permits designers to select properties to be verified and to specify additional security policies. Once designers are confident with the model, the resulting Datalog specification is verified by Datalog solvers with respect to the properties that designers want to check. Then, the solver output is parsed by DF in order to present it in a more user-readable format.

Chapter 5

Conclusion

We have presented a formal framework for reasoning with goal models developed during agent-oriented software development using the Tropos methodology. Our work combines earlier works on an agent-oriented software methodology [BGG$^+$04a, CKM02], with a formal goal model defined and studied in [GNMR03, RGM04]. In a parallel effort [FLP$^+$03] we have developed a tool, called the T-tool, which supports temporal reasoning of formal Tropos specifications using a state-of-the-art model checker. Together, these tools can help the developer of Tropos models analyze them and make sure they are consistent with her intuitions. This feedback is essential, especially so when one is dealing with large models developed by a team of designers.

We envision other types of formal analysis for Tropos models. In particular, we propose to work on formal actor dependency models and develop scalable – and usable social analysis techniques that complement the temporal and intentional analysis techniques developed so far.

Then, we have presented a new semantic schema matching algorithm and its optimizations. Our solution builds on the top of the past approaches at the element level and introduces a novel (with respect to schema matching) techniques, namely model-based techniques, at the structure level. We conducted a comparative evaluation of our approach implemented in the S-Match system against three state of the art systems. The results empirically prove the strengths of our approach.

Future work along this line includes development of the *iterative* and *interactive* semantic matching system. It will improve the quality of the mappings by iterating and by focusing user's attention on the critical points where his/her input is maximally useful. S-Match works in a top-down manner, and hence, mismatches among the top level elements of schemas can imply further mismatches between their descendants. Therefore, next steps include development of a *robust* semantic matching algorithm. Finally, we are going to develop a *testing methodology* which is able to estimate quality of the mappings between schemas with hundreds and thousands of nodes. Initial steps have already been done, see for details [AGY05]. Here, the key issue is that in these cases, specifying reference mappings manually is (often) neither desirable nor feasible task. Comparison of matching algorithms on real-world schemas from different application domains will also be performed more *extensively*.

Finally, we have presented the ST-Tool, a CASE Tool for Security Requirements Engineering. We have already used the tool to model a comprehensive case study on the compliance to the Italian legislation on Privacy and Data Protection by the University of Trento, leading to the definition and analysis of an ISO-17799-like security management scheme [MPZ05].

Future work will involve a front-end with T-Tool [FLP^{+}03] for automatically verifying Formal Tropos specification. Further, Secure Tropos is still under work, so is ST-Tool, too. We are also considering to integrate our tools into the ECLIPSE platform.

Chapter 6

History of the Deliverable

Below we outline how the work described in the deliverable has evolved along the years of the project.

Change History

Version	Date	Status	Author (Unit)	Description
0.1	2004/Nov	Draft	UNITN	First version of the goal-oriented requirements analysis and reasoning in the Tropos methodology.
0.2	2005/Nov	Draft	UNITN	First version of semantic matching for goal diagrams.
0.3	2005/Nov	Draft	UNITN	First version of the security requirements engineering with ST-Tool.
1.0	2006/Nov	Final	UNITN	Final revision of the deliverable.

Bibliography

[AGY05] P. Avesani, F. Giunchiglia, and M. Yatskevich. A large scale taxonomy mapping evaluation. In *Proceedings of ISWC*, 2005.

[BCV99] S. Bergamaschi, S. Castano, and M. Vincini. Semantic integration of semistructured and structured data sources. *SIGMOD Record*, pages 54–59, 1999.

[Ber] D. Le Berre. A satisfiability library for Java. http://www.sat4j.org/.

[BGG$^+$04a] P. Bresciani, P. Giorgini, F. Giunchiglia, J. Mylopoulos, and A. Perini. Tropos: An agent-oriented software development methodology. *Journal of Autonomous Agents and Multi-Agent Systems*, 8(3):203–236, May 2004.

[BGG$^+$04b] Paolo Bresciani, Paolo Giorgini, Fausto Giunchiglia, John Mylopoulos, and Anna Perini. TROPOS: An Agent-Oriented Software Development Methodology. *JAAMAS*, 8(3):203–236, 2004.

[BMPQ04] P. Bernstein, S. Melnik, M. Petropoulos, and C. Quix. Industrial-strength schema matching. *SIGMOD Record*, 33(4):38 – 43, 2004.

[BSZ03] P. Bouquet, L. Serafini, and S. Zanobini. Semantic coordination: A new approach and an application. In *Proceedings of ISWC*, pages 130–145, 2003.

[CKM02] Jaelson Castro, Manuel Kolp, and John Mylopoulos. Towards Requirements-Driven Information Systems Engineering: The Tropos Project. *Information Systems*, 27(6):365–389, 2002.

[Con00] J. Conallen. *Building Web Applications with UML*. Addison-Wesley, 2000.

[DLD$^+$04] R. Dhamankar, Y. Lee, A. Doan, A. Halevy, and P. Domingos. iMAP: Discovering complex semantic matches between database schemas. In *Proceedings of SIGMOD*, pages 383–394, 2004.

[DLL62] M. Davis, G. Longemann, and D. Loveland. A machine program for theorem proving. *Journal of the ACM*, (5(7)):394–397, 1962.

[DR02] H. H. Do and E. Rahm. COMA - a system for flexible combination of schema matching approaches. In *Proceedings of VLDB*, pages 610–621, 2002.

[DvLF93a] A. Dardenne, A. van Lamsweerde, and S. Fickas. Goal-directed requirements acquisition. *Science of Computer Programming*, 20(1–2):3–50, 1993.

[DvLF93b] A. Dardenne, A. van Lamsweerde, and S. Fickas. Goal-directed requirements acquisition. *Science of Computer Programming*, 20(1–2):3–50, 1993.

[ea06] P. Bresciani et al. KLASE MIUR-FIRB project RBAU01P5SS "Knowledge Level Automated Software Engineering", WP1.2. Technical report, University of Trento, 2006.

[EV04] J. Euzenat and P. Valtchev. Similarity-based ontology alignment in OWL-lite. In *Proceedings of ECAI*, pages 333–337, 2004.

[FLP$^+$03] Ariel Fuxman, Lin Liu, Marco Pistore, Marco Roveri, and John Mylopoulos. Specifying and analyzing early requirements: Some experimental results. In *Proceedings of the 11th IEEE International Requirements Engineering Conference (RE'03)*, page 105. IEEE Computer Society Press, 2003.

[GATTM05] A. Gal, A. Anaby-Tavor, A. Trombetta, and D. Montesi. A framework for modeling and evaluating automatic semantic reconciliation. *VLDB Journal*, (14(1)), 2005.

[GMM$^+$05] P. Giorgini, F. Massacci, J. Mylopoulos, A. Siena, and N. Zannone. St-tool: a case tool for modeling and analyzing trust requirements. In Herrmann P., Issarny V., and Shiu S., editors, *3rd international conference on trust management (iTrust 2005)*, volume 3477 of *Lecture notes in computer science*, pages 415–419, Rocquencourt, France, 23-26 May 2005. Springer.

[GMMZ04] Paolo Giorgini, Fabio Massacci, John Mylopoulos, and Nicola Zannone. Requirements Engineering meets Trust Management: Model, Methodology, and Reasoning. In *Proc. of iTrust'04*, volume 2995 of *LNCS*, pages 176–190. Springer-Verlag, 2004.

[GMMZ05] P. Giorgini, F. Massacci, J. Mylopoulos, and N. Zannone. St-tool: A case tool for security requirements engineering. In *Proceedings of the13th IEEE International Conference on Requirements Engineering*, pages 451–452, Paris, France, 29 August - 2 September 2005. IEEE Computer Society.

[GMS04] P. Giorgini, J. Mylopoulos, and R. Sebastiani. Goal-Oriented Requirements Analysis and Reasoning in the Tropos Methodology. *Engineering Application of Artificial Intelligence Journal*, 18(2), 2004.

[GNMR02] P. Giorgini, E. Nicchiarelli, J. Mylopoulos, and R.Sebastiani. Reasoning with Goal Models. In *Proc. Int. Conference of Conceptual Modeling – ER'02*, volume 2503 of *LNCS*, Tampere, Finland, October 2002. Springer.

[GNMR03] P. Giorgini, E. Nicchiarelli, J. Mylopoulos, and R.Sebastiani. Formal Reasoning Techniques for Goal Models. *Journal of Data Semantics*, 1, October 2003. Springer.

[GNMS02] P. Giorgini, E. Nicchiarelli, J. Mylopoulos, and R. Sebastiani. Reasoning with Goal Models. In *Proc. Int. Conference of Conceptual Modeling – ER'02*, LNCS, Tampere, Finland, October 2002. Springer.

[GNMS04] P. Giorgini, E. Nicchiarelli, J. Mylopoulos, and R. Sebastiani. Formal Reasoning Techniques for Goal Models. *Journal of Data Semantics*, 1, October 2004. Springer.

[GS03a] F. Giunchiglia and P. Shvaiko. Semantic matching. *The Knowledge Engineering Review*, 18(3):265–280, 2003.

[GS03b] F. Giunchiglia and P. Shvaiko. Semantic matching. *KER Journal*, (18(3)), 2003.

[GSY04] F. Giunchiglia, P. Shvaiko, and M. Yatskevich. S-match: An algorithm and implementation of semantic matching. In *Proceedings of ESWS'04*, pages 61–75, 2004.

[GSY05] F. Giunchiglia, P. Shvaiko, and M. Yatskevich. Semantic schema matching. In *Proceedings of CoopIS*, pages 347–365, 2005.

[GSY06] F. Giunchiglia, P. Shvaiko, and M. Yatskevich. Discovering missing background knowledge in ontology matching. In *Proceedings of ECAI*, 2006.

[Gua04] N. Guarino. The role of ontologies for the Semantic Web (and beyond). Technical report, Laboratory for Applied Ontology, ISTC-CNR, 2004.

[GY04] F. Giunchiglia and M. Yatskevich. Element level semantic matching. In *In Proceedings of Meaning Coordination and Negotiation workshop at ISWC'04*, 2004.

[HMW] V. Haarslev, R. Moller, and M. Wessel. RACER: Semantic middleware for industrial projects based on RDF/OWL, a W3C Standard. `http://www.sts.tu-harburg.de/~r.f.moeller/racer/`.

[KN03] J. Kang and J. F. Naughton. On schema matching with opaque column names and data values. In *Proceedings of SIGMOD*, pages 205–216, 2003.

[Lib00] P. Liberatore. Algorithms and Experiments on Finding Minimal Models. Technical report, DIS, University of Rome "La Sapienza", December 2000. Available at http://www.dis.uniroma1.it/˜liberato/mindp/.

[MBR01] J. Madhavan, P. Bernstein, and E. Rahm. Generic schema matching with Cupid. In *Proceedings of VLDB*, pages 49–58, 2001.

[MCN92] J. Mylopoulos, L. Chung, and B. Nixon. Representing and Using Non-Functional Requirements: A Process-Oriented Approach. *IEEE Transactions on Software Engineering*, 6(18):483–497, June 1992.

[MGMR02] S. Melnik, H. Garcia-Molina, and E. Rahm. Similarity flooding: A versatile graph matching algorithm. In *Proceedings of ICDE*, pages 117–128, 2002.

[Mil95] A. G. Miller. WordNet: A lexical database for English. *Communications of the ACM*, (38(11)):39–41, 1995.

[MMZ$^+$01] M. W. Moskewicz, C. F. Madigan, Y. Z., L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In *Design Automation Conference*, 2001.

[MPZ05] F. Massacci, M. Prest, and N. Zannone. Using a security requirements engineering methodology in practice: the compliance with the italian data protection legislation. *Computer Standards & Interfaces*, 27(5):445–455, 2005.

[MRB03] S. Melnik, E. Rahm, and P. Bernstein. Rondo: A programming platform for generic model management. In *Proceedings of SIGMOD*, pages 193–204, 2003.

[MSGPdS04] D. L. McGuinness, P. Shvaiko, F. Giunchiglia, and P. Pinheiro da Silva. Towards explaining semantic matching. In *In Proceedings of international workshop on DL at KR'04*, 2004.

[MSS03] B. Magnini, L. Serafini, and M. Speranza. Making explicit the semantics hidden in schema models. In *Proceedings of workshop on HLTSWWS at ISWC*, 2003.

[NS] A. Newell and H. Simon. GPS: A Program that Simulates Human Thought. In E. Feigenbaum and J. Feldman, editors, *Computers and Thought*. McGraw Hill.

[Pan04] J. Z. Pan. *Description Logics: reasoning support for the Semantic Web*. PhD thesis, School of Computer Science, The University of Manchester, 2004.

[PG86] D. Plaisted and S. Greenbaum. A structure-preserving clause form translation. *Journal of Symbolic Computation*, (2):293–304, 1986.

[RB01] E. Rahm and P. Bernstein. A survey of approaches to automatic schema matching. *VLDB Journal*, (10(4)):334–350, 2001.

[RGM04] R.Sebastiani, P. Giorgini, and J. Mylopoulos. Simple and Minimum-Cost Satisfiability for Goal Models. In *Proc. Int. conference on Advanced Information Systems Engineering, CAISE'04*, volume 3084 of *LNCS*, pages 20–33. Springer, June 2004.

[Rol03] C. Rolland. Reasoning with Goals to Engineer Requirements. In *Proceedings 5th International Conference on Enterprise Information Systems*, 2003.

[SE05] P. Shvaiko and J. Euzenat. A survey of schema-based matching approaches. *Journal on Data Semantics*, IV, 2005.

[SWM04] M. K. Smith, C. Welty, and D. L. McGuinness. OWL web ontology language guide. Technical report, World Wide Web Consortium (W3C), http://www.w3.org/TR/2004/REC-owl-guide-20040210/, February 10 2004.

[vL00] A. v. Lamsweerde. Requirements engineering in the year 00: A research perspective. In *Proceedings 22nd International Conference on Software Engineering, Invited Paper, ACM Press*, 2000.

[Yu95] E. Yu. *Modelling Strategic Relationships for Process Reengineering*. PhD thesis, University of Toronto, Department of Computer Science, 1995.

[ZM02] Lintao Zhang and Sharad Malik. The quest for efficient boolean satisfiability solvers. In *Proc. CAV'02*, number 2404 in LNCS, pages 17–36. Springer, 2002.