TERENCE project – ICT FP7 Programme – ICT-2010-257410

Reasoning Module and Textual Smart Game Repository

Document Version: *9*
Created: *August 28, 2013*
Last updated: *August 28, 2013*
Distribution level: *Public*

Document Information

ID	D4.2,D4.3		
Keywords	qualitative temporal representation and reasoning, TimeML, semantic web, natural language generation, constraint-based reasoning, automated reasoning, heuristics, game based learning, TEL		
Classification	*Version* 9	*Date of reference* August 30, 2013	*Published date* August 28, 2013
Distribution level	Public		

Distribution level:

- **Internal**: Circulation is restricted to the partners of the TERENCE Consortium
- **Restricted**: Circulation is restricted to the partners of the TERENCE Consortium and the EU Commission for project evaluation purposes
- **Public**: This document may be distributed to the general public for dissemination and information purposes

	Surname and Name	**Company**
Editor	Rosella Gennari	LUB
List of contributors	Mohammad Alrifai	LUH
	Gianni Barlacchi	LUB
	Giang Binh Tran	LUH
	Fernando De la Prieta Pintado	USAL
	Rosella Gennari	LUB
	Alessio Paolucci	LUB, UnivAQ
	Oana Tifrea	LUB
	Sara Tonelli	FBK-irst
	Pierpaolo Vittorini	UnivAQ

Version history		
Date	*Version*	*Description*
29/11/2011	1	Structure and initial content
02/12/2011	2	Refinements in light of the current module implementation
05/12/2011	3	First release
08/12/2011	4	Structure reorganisation and description of `tml2gqr` library
01/01/2012	5	First update due to the move from WSDL to REST
01/05/2012	6	Second update due to changes in the annotation language of D3.1
30/05/2012	7	Final first release
29/01/2013	8	Second release
28/08/2013	9	Third release

Abstract

TERENCE is a Collaborative project funded by the European Commission and developing the first intelligent adaptive learning system for poor text comprehenders from primary schools and for their educators. Its learning material is made of stories, collected into books, and smart games for reasoning about stories. One of the ambitions of TERENCE is to generate textual components for such games *automatically*, starting from flat stories, in Italian and in English. To this aim, stories are first analysed and annotated in work package 3 of the project with a specific mark-up annotation language. This report shows how the automated generation process takes place in work package 4 from such annotated stories. The report pertains to two deliverables this work package, namely, D4.2 and D4.3. In a nutshell, the report shows how D4.2 develops a *reasoning module* for generating textual components of smart game instances, stored in D4.3, by combining (1) *temporal constraint algorithms and tools* with (2) *natural language generation algorithms and tools*.

The generation process as well as the entire TERENCE system were first designed after analysing the context of use of work package 1 of the project. Afterwards, the evaluations of work package 7 were used to revise the design, iteratively and incrementally. In particular, the second expert-based evaluation of smart games reported in [12] assessed the grammatical correctness, plausibility and relevance of the automatically generated textual components of smart games of the second release of D4.2 and D4.3. Results of this evaluation were analysed and used to revise some features of the generation process design for the third releases, as outlined in end of this report.

The follow-up executive summary outlines the contents, novelties and structure of this report for the final releases of D4.2 and D4.3.

Copyright notices

Contents

List of Figures

List of Tables

1 Executive summary

1.1 Motivations

From the age of 7–8 until the age of 11, children develop as independent readers. Nowadays, more and more children in that age range turn out to be poor text comprehenders: they demonstrate difficulties in deep text comprehension, despite well developed low-level cognitive skills such as vocabulary knowledge, e.g., see [36]. TERENCE is a Collaborative project funded by the European Commission and developing the first intelligent *adaptive learning system* (ALS) [33] for poor text comprehenders from primary schools, that is, the TERENCE *learners*, and their educators. The system proposes stories, organised into difficulty categories and collected into books, and smart games for reasoning about events of a story and their relations.

The TERENCE smart games are serious games [21] and, like the entire ALS, are designed within a stimulation plan for the TERENCE learners. One of the goals of TERENCE is to generate textual components for games semi-automatically, starting from flat stories for primary-school children, in Italian and in English, using artificial-intelligence technologies and tools. The overall ambition is to see whether and how that can help to improve development performances, given that we deal with 256 different stories, each having c. 12 games. To this end, stories are annotated with natural language processing techniques in work package 3 (WP3) of the project. This report shows how the automated generation process takes place from such annotations.

This report pertains to two deliverables of work package 4 (WP4), namely, D4.2 and D4.3. The report shows how D4.2 develops a *reasoning module* for generating textual components of smart game instances, stored in D4.3, by combining

- *temporal constraint* (TC) algorithms and tools
- *natural language generation* (NLG) techniques and tools.

The design of the generation process, and of the entire TERENCE system in general, is rooted in the context of use analysis run at the start of the TERENCE project, and has been iteratively and incrementally revised in light of the evaluation results of the project.

1.2 The Focus of the Report

The starting point for such a generation process is a mark-up language, the TERENCE annotation language, that allows for the annotation of stories' key features for the smart games for the TERENCE learners. The TERENCE annotation language is used by the annotation module for Italian and English of WP3 to tag key features of stories, such as factual events and consecutive temporal relations.

In turn, the annotations returned by WP3 are used by the reasoning module (D4.2) of WP4 with two goals:

G1. reasoning about the annotations of stories stored in D3.3 of WP3 in order to produce enriched annotated stories, and hence

G2. reasoning about these stories in order to automatically generate textual components of smart games, ranked according to their relevance for the TERENCE learners.

As for G1, the reasoning module uses TC technologies to enhance the annotation process of stories performed in WP3: for consistency checking of temporal annotations with respect to their formal semantics over reals; for deduction of relations, that are left out during the annotation process of WP3. In this manner, the module complements the annotation process over stories performed in WP3.

As for G2, the reasoning module combines NLG and TC algorithms to generate textual components of smart game instances, *textual games*, in brief. It does it automatically for each game level, using the pertinent framework, as specified in [4].

Figure 1.1 recaps the automated generation process. More precisely, the figure shows the overall workflow of processes and data starting from (1) the annotation of flat stories made by the annotation module of WP3, (2) the consistency checking and enrichment of the annotated stories performed by D4.2, (3) the generation of textual components of smart games performed by D4.3 and stored in D4.3. Finally the visualisation module concludes the process by assembling the textual and graphical components of the entire learning material, including smart games, in the learner GUI. For this, see [42].

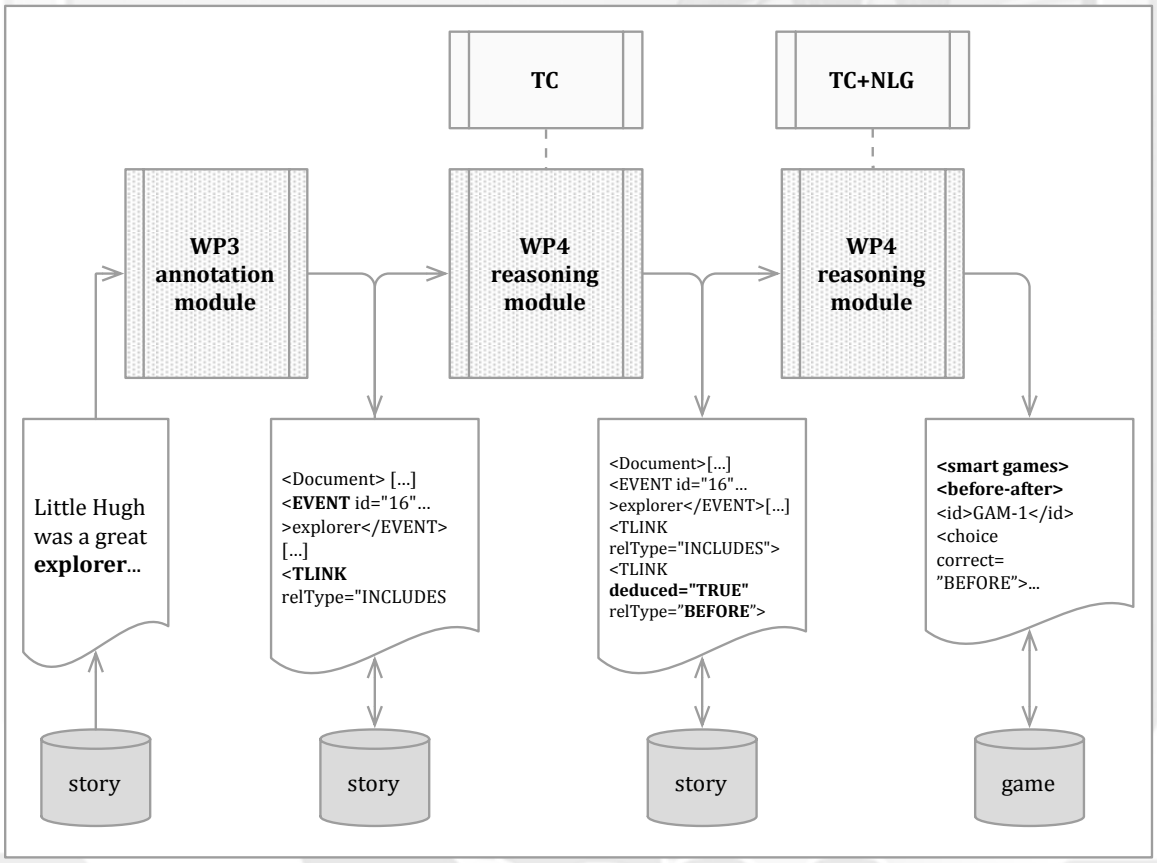


Figure 1.1: The automated generation process of textual components of smart games in XML

1.3 Outline of the Report and Main Changes

This TR is divided in the following main chapters.

Chapter 2 is a ***brand new chapter*** with background information recapped and restyled from other deliverables for the aims of this report. Section 2.1 specifies the essential features of the TERENCE annotation language necessary for generating textual components of smart games. Section 2.2 is about the chosen semantics for the TERENCE annotation language and is based on [17], to which we refer for more detailed information on qualitative temporal representation and reasoning. The semantics allows the reasoning module to check the consistency of the annotations and deduce further knowledge for the generation of games. Section 2.3 concludes Chapter 2 and recaps the essential features of the design of smart games and, as such, is based on [4]. In particular, this section outlines those elements of the TERENCE smart game design that allow the reader to understand the automated generation process.

Sections 2.1 and 2.2 are necessary prerequisites for reading Chapter 3, which shows how the aforementioned annotations with their formal semantics are put to work in order to 'give meaning' to the stories' annotations, so as to check the consistency of temporal annotations and deduce further relations.

The reading of this chapter facilitates the understanding of the remaining part of the report but is not necessary to it. Sections 2.1 and 2.3, instead, are necessary prerequisites for Chapters 4, 5 and 6.

In fact, Chapter 4 outlines the process of generation of textual instances of smart games, and gives examples of instances so generated. ***This chapter is totally revised with respect to the previous version of this report.***

Chapter 5 delves into the natural language generation processes, for Italian and English, of specific textual components of smart games, which are language dependent. ***This chapter contains minor changes due to optimisation in the modules, in particular, the English one.***

Chapter 6 contains ***novel NLG work*** for the generation of more plausible wrong solutions, in light of the results of the second expert-based evaluation of TERENCE, reported in [12].

Figure 1.2 recaps the reading dependencies of chapters or sections. Totally new material is coloured in yellow. Grey-shaded chapters contain minor differences with respect to the second release due to optimisations.

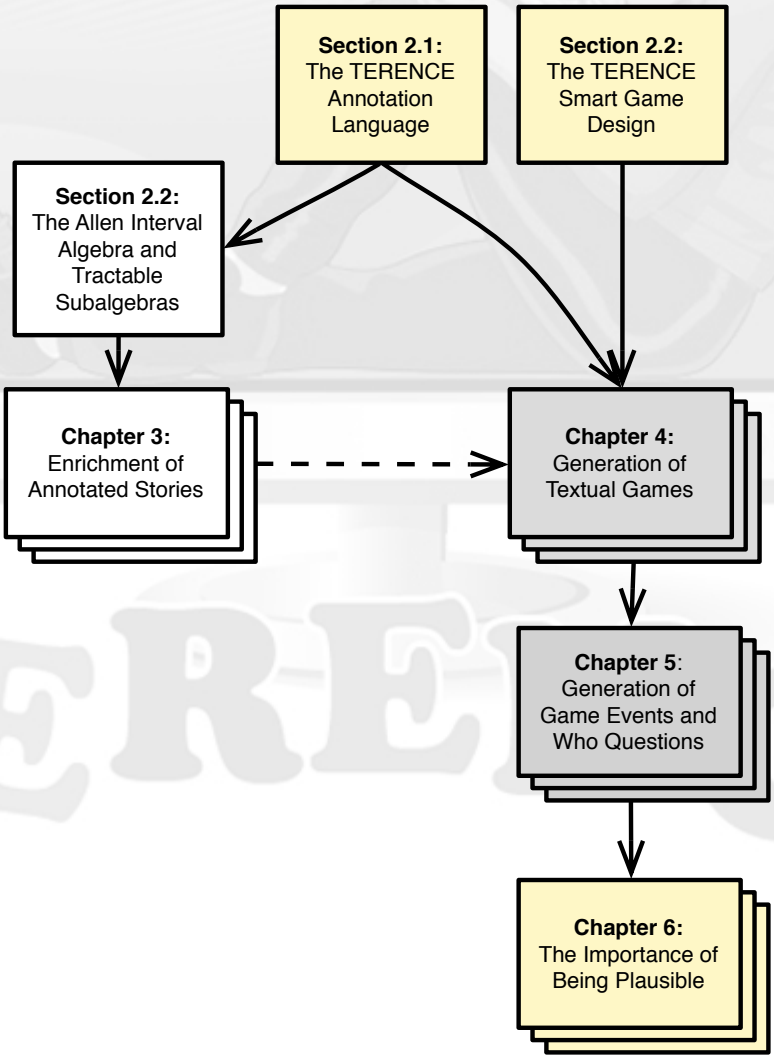


Figure 1.2: Reading dependencies

2 Background

2.1 The TERENCE Annotation Language

The events of a story, their temporal and causal-temporal relations, and the characters that participate in events are at the heart of the TERENCE smart games. Therefore TERENCE needs an annotation language that can specify them. TimeML [29, 31] is a good starting point for that: it covers events and qualitative temporal information, relevant for stories, and is the de-facto standard markup language for temporal information in the *natural language processing* (NLP) community, which allows for the re-use of existing NLP and TC tools for qualitative temporal reasoning. See Subsection 2.1.1 for an introduction to TimeML. The analysis of the limitations lies at the heart of the TERENCE annotation language, built on top of TimeML, as outlined in Subsection 2.1.2.

2.1.1 The TimeML Annotation Format

TimeML [29, 31] is an international markup language for annotating events and temporal information in a text. It was designed in order to address the following four issues: *(i)* identify events and anchor them in time, *(ii)* order events, *(iii)* reason with contextually underspecified temporal expressions and *(iv)* reason about the persistence of events. TimeML includes three major data structures, namely, a qualitative time entity (e.g., action verbs) called `EVENT`, a quantitative time entity (e.g., dates) called `TIMEX`, and relations among them called generically links. In particular, in TimeML events can be expressed by tensed and untensed verbs, but also by nominalizations (e.g. *invasion*, *discussion*, *speech*), predicative clauses (e.g. *to be the President of something*), adjectives (e.g. *dormant*) or prepositional phrases (e.g. *on board*). TimeML is by now a standard in the NLP community and is used in the annotation of linguistic resources such as the TimeBank corpus [30], the Ita-TimeBank [10] and of news events in the 2010 TempEval competition [41].

As for `TIMEX` temporal expressions, they include specific dates (e.g. *June 11, 1989*), and durations (*three months*). TimeML includes also the concept of time normalisation, where a normalised form is assigned to each expression based on consistency and interchange format in line with the ISO 8601 standard for representing dates, times, and durations. Time entities (`EVENT` and `TIMEX`) can be connected through a qualitative temporal relation called `TLINK`, making explicit it the two entities are simultaneous, identical or in sequential order. Specifically, several types of relations have been introduced in TimeML modelled on *Allen interval algebra* of qualitative temporal relations [2]. Other types of links are also present in TimeML standard, for instance subordination links between specific verbs of reporting and opinion and their event arguments, or aspectual links between aspectual events and their argument events. However, since they are not included in the TERENCE framework, we do not discuss them in details.

2.1.2 The TERENCE Annotation Format (TAF)

TimeML is the de-facto standard language on which the NLP communities have grown their experience and tools in the last decade. However, TimeML is sometimes underspecified and sometimes too detailed for the TERENCE purposes. For instance, the semantics of `TLINKs` in terms of Allen relations is not uniquely specified but, in the working practice, is pretty context dependent. Key information like the participants in an event have already been proposed [32] but neither their attributes nor their relations with events have been clearly

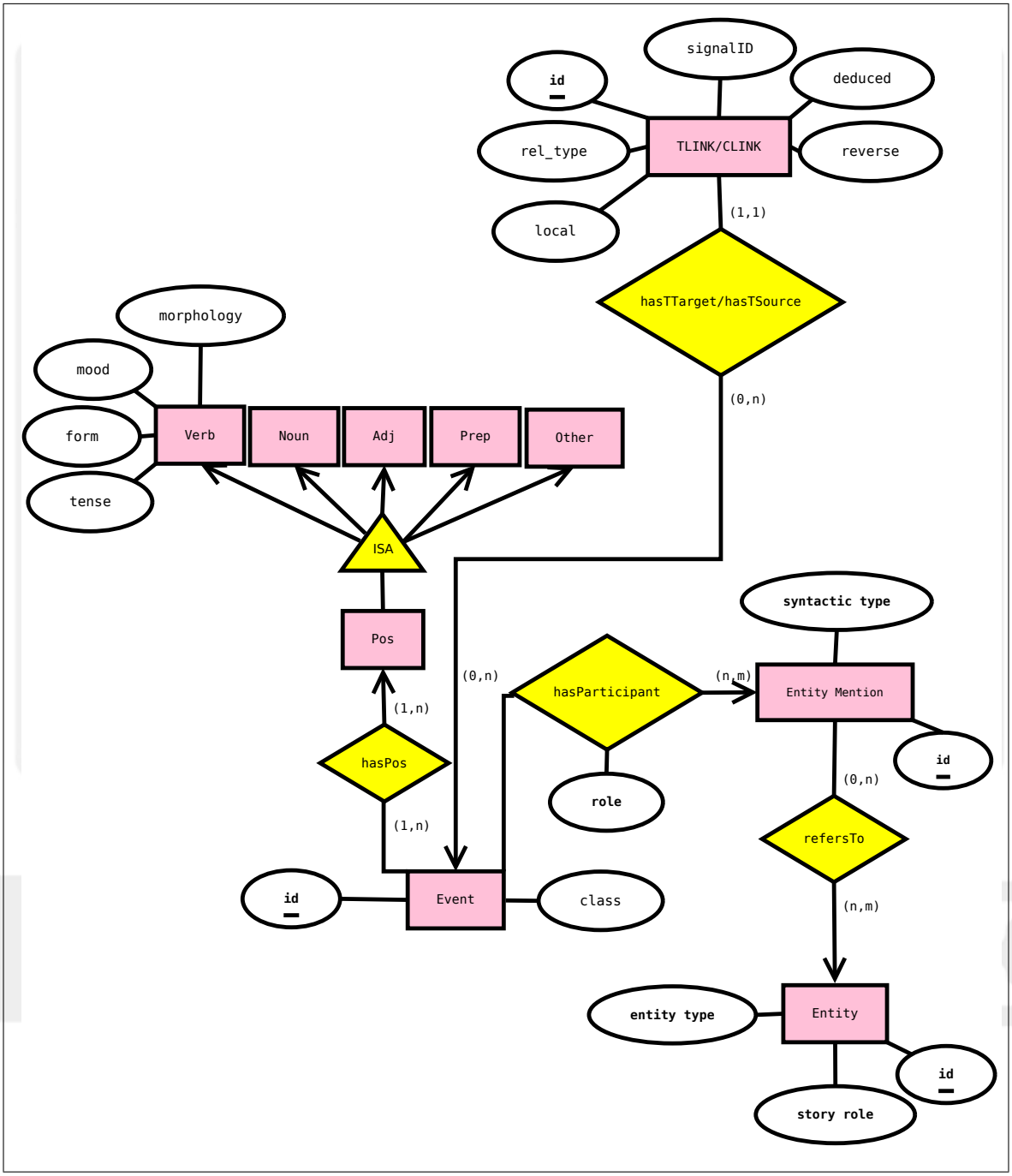


Figure 2.1: EER of events and their relations

pinpointed. Since knowing which the characters in a story are, their role and how they interact is crucial for the TERENCE smart games, such information is essential for analysing the TERENCE stories. Another main problem for TERENCE with the TimeML language is the lack of causal links when explicitly signalled by linguistic cues such as “because" and “since". In several other cases, the level of detail foreseen in TimeML is not needed to process children’s stories such as in TERENCE. For instance, while in TimeML modal and hypothetical actions are annotated as events, the primary interest of TERENCE are the restricted events that actually take place in the story, namely, so-called factual events.

To take advantage of the pros of TimeML and cope with its limitations, we built the *TERENCE Annotation Format* (TAF) on top of TimeML. See [9]. Hereby we outline the main design choices of the language, relevant for generating diverse smart games, their wrong and correct solutions, as well as for classifying smart games into fine-grained difficulty categories. The requirements for the language and the analyses leading to such requirements are in [36]. For instance, well-placed signals like connectives tend to easy their understanding of relations between events. Thus a requirement of TAF is to trace whether, for instance, a temporal or causal-temporal relation between events is rendered by a connective, and if the events occur in the same sentence, or otherwise, keeping track of where.

For explaining TAF, we use the *Extended Entity-Relationship* (EER) [38] diagram in Figure 2.1. Therein, classes are coloured in pink. Relations among classes are yellow. Attributes are in balloons. Each class has a unique key identifying attribute. Cardinality constraints between an entity and an attribute are equal to $(1, 1)$ unless differently set. The diagram in Figure 2.1 specifies the classes of `Event` and `Pos`, denoting an event and the lemma of the word evoking it, respectively. The attributes of `Pos` specify relevant information, in particular, if it is a verb and, e.g., its regularity. In the diagram in Figure 2.1, we also have `TIMEX` and `TLINKs`. Since children’s stories, like in TERENCE, are not usually anchored to a specific date, time normalisation is often not possible, therefore temporal expressions are not normalised as opposed to TimeML standard. Therefore we focus on the analysis of a `TLINK`, which is relevant for time games. A `TLINK` denotes a temporal binary relation between events, and has the following attributes:

1. `reverse`, with Booleans “true” or “false”, capturing if the order of occurrence of the two events in the story text is the temporal order (“false”) or not (“true”);
2. `signalID`, with string values the id of the annotated signal (e.g., “30” for the signal “and then”) if the two `Events` are related through an explicit temporal signal, or “none” otherwise;
3. `rel_type`, with string value the TimeML relations “before”, “after”, “overlaps”, “is_included”, “includes”, “identity”, “all”;
4. `local`, with Boolean values “true” if the two `Events` are in the same sentence or in consecutive sentences, or “false” otherwise.

Except for `rel_type`, such attributes are not present in TimeML. The `hasTTarget` and `hasTSource` relations serve to specify, respectively, the source event and the target event of the relation, which is done differently than in TimeML. Similar remarks apply for the `CLINK` class, its attributes, the `hasTTarget` and `hasTSource` relations. In particular, in TAF and not in TimeML we have causal links between events when explicitly signaled by linguistic cues such as “because", “since". The `CLINKs` are used for causality games.

The diagram in Figure 2.1 also introduces classes and relations for specifying the characters or other entities participating in events of a story, as well as their role within the story. In particular, `Entity` refers to the unique identifier of a participant, like a character in a story (e.g., the proper name “Ben” of a male character), whereas the `Entity Mention` stands for any expression that correlates via `refersTo` to a unique `Entity` (e.g., “the boy”); such a distinction between entities and their story mentions has been introduced to account for coreference in TAF, which is not covered in TimeML. The relation `hasParticipant` correlates each `Event` to all the `Entity Mentions` involved in such event. The relation has the composite attribute named `role`, which stands in for the role that a correlated entity mention has in the event. One of its atomic attributes is `semantic role`, and among its values we have “agent” for the agents of an event. While `Entity Mentions` are described through the syntactic type they have in the sentence in which they occur (e.g. “subject”, “object”). `Entities` are characterised at the semantic level with the following attributes with string values:

1. `entity type`, with values “person”, “location”, “animal”, “pseudo-person”, “vegetable”, “artefact”, “other”;
2. `story role`, with values “protagonist”, “secondary”, “minor”.

Such information is relevant for the generation of more plausible wrong solutions in games and for organising them into fine-grained difficulty categories.

2.2 The Allen Interval Algebra and Tractable Subalgebras

The TimeML language is based on Allen relations [31], and so is the derived TAF mark-up language described above. Neither one language nor the other comes with pre-defined semantics for its entities in terms of Allen relations, being the choice of the semantics rather context-dependent. However, in order to decide whether the temporal flow of an annotated story 'makes sense', we need to assign a semantics to TAF temporal entities: if we map tags into Allen relations, we provide them with a semantics, and can then automatically reason with it, for instance, we can decide about the temporal consistency of the story as well as deduce further temporal relations between its events. In order to understand how to assign such a semantics and use it for performing consistency checking and deduction, automatically, we need first some knowledge of what the Allen relation algebra and sub-algebras are. That is precisely the goal of this section. We start with sketching what the entire Allen algebra is, the importance of its so-called tractable subalgebras and end up explaining what it means to reason with them.

2.2.1 Algebras and subalgebras

In his seminal paper [2], Allen motivated his time representation as follows: "This representation is designed explicitly to deal with the problem that much of our temporal knowledge is relative, and hence cannot be described by a date (or even a fuzzy date)". In the Allen representation, the only events are time intervals. Between any two pairs of events, there is precisely one *atomic Allen relation*, namely, a relation at of the form before, meets, during, overlaps, starts, during, finishes, equals, or its inverse at^{-1}. See Figure 2.2 for their (standard) interval interpretation, not naming the starting and ending points of intervals.

As Allen arguments, his representation of time allows for "significant imprecision" whenever, as in the TERENCE stories, it is often the case that temporal knowledge is relative without relations to absolute dates. Indefinite information can be represented by means of disjunctions (unions) of the Allen atomic relations through $\vee$. Then an *Allen relation* rel is a disjunction of atomic relations. An example is before $\vee$ meets.

The set of Allen relations forms the *Allen Interval Algebra* (IA) with conjunction (intersection), inverse $^{-1}$ and composition $\bowtie$.

Note that disjunctions of TLINKS relations are not foreseen in TimeML nor in TAF. This can be rather restrictive when annotating stories for children, due to inherent imprecision of data (e.g., "Early in the afternoon, Perla went to the library") or different text interpretations by the annotators (e.g., knowledge dependent information). Therefore, in this setting, one may need a semantics that maps TLINKS into disjunctive Allen relations to account for such fuzziness.

One could use relations of a subalgebra of the Allen one, say, the *continuous-endpoint subalgebra* (CA), that is computationally tractable—we will specify what we mean by a tractable subalgebra below, after introducing the necessary details. This is a widely investigated subalgebra, already used in several applications of NLP to narratives [40], that allows for expressing vague information such as `before` $\vee$ `meets` and `before` $\vee$ `meets` $\vee$ `overlaps`.

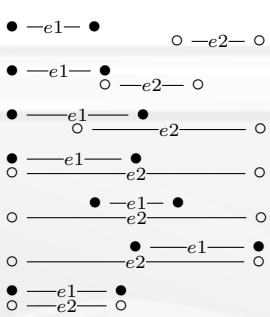
2.2.2 Allen temporal constraint problems and algorithms

In order to automatically reason with Allen relations, we use constraint problems and algorithms, as explained in this subsection. Let A be a subset of the Allen interval algebra, of qualitative relations between events. An *(Allen binary) constraint (satisfaction) problem* P over A is given by

- a finite sequence of variables, $e_1, e_2, \ldots, e_n$, where each e_i represents an event and ranges over a finite collection D_i of intervals of reals,

– and a (binary) constraint $C(i,j) \in A$ for each pair of variables (e_i, e_j) with $0 \leq i < j \leq n$.

This is the classical encoding of a qualitative temporal (satisfiability) problem with Allen relations into a constraint problem, albeit others are possible, e.g., see [5].



$e1$ before $e2$, $e2$ before^{-1} $e1$
$e1$ meets $e2$, $e2$ meets^{-1} $e1$
$e1$ overlaps $e2$, $e2$ overlaps^{-1} $e1$
$e1$ starts $e2$, $e2$ starts^{-1} $e1$
$e1$ during $e2$, $e2$ during^{-1} $e1$
$e1$ finishes $e2$, $e2$ finishes^{-1} $e1$
$e1$ equals $e2$, $e2$ equals $e1$

Figure 2.2: The atomic Allen relations.

In the following, without sacrificing generality, we will assume that there is precisely one constraint for each pair of events e_i and e_j, with $i < j$. In the literature, this means that our constraint problems are normalised, see [5].

With a slight abuse of notation, we will write $I_i\, C(i,j) I_j$ if (I_i, I_j) is an interval interpretation (as in Figure 2.2) of one of the atomic disjuncts of $C(i,j)$. A tuple of intervals $(I_1, \ldots, I_n)$ of $D_1 \times \cdots \times D_n$ is a *solution* to the constraint problem P if $I_i\, C(i,j) I_j$ for each $C(i,j)$ of P. More generally, P is *satisfiable* or *consistent* if it has a solution, *unsatisfiable* or *inconsistent* otherwise.

Example 1 *Consider the constraint problem with two events, namely, e_1 and e_2, and constraint $C(1,2)$ equal to* `before`$\vee$`meets`*. A solution to this constraint problem is the pair $([0,1],[2,3])$. Another solution is $([0,1],[1,2])$. This problem is thus consistent. The problem with events e_1, e_2 and e_3, constraints $C(1,2)$ equal to* `before` $\vee$ `meets`*, $C(1,3)$ equal to* `after`*, and $C(2,3)$ equal to* `before` *is inconsistent.*

We will say that $rel \in A$ for (e_i, e_j) is *deduced* if $I_i\, rel\, I_j$ holds for all solutions $(I_1, \ldots, I_n)$ to P. Let DC_{ij} be the set of all the deduced relations for (e_i, e_j). The *deductive closure* of P is the set of all such DC_{ij}, for $0 \leq i < j \leq n$. Clearly, a problem is consistent if and only if its deductive closure is different than the empty set.

Example 2 *Consider the constraint problem with three events, namely, e_1, e_2 and e_3, constraints $C(1,2)$ and $C(1,3)$ equal to* `before` $\vee$ `meets`*, whereas $C(2,3)$ is equal to* `before`*. The deductive closure of this problem has* `before` $\vee$ `meets` *for e_1 and e_2, and* `before` *for e_1 and e_3 as well as for e_2 and e_3.*

If there is a PTIME algorithm that can decide about the satisfiability of any problem over A, then we say that A is a *tractable subalgebra*. In case the tractable subalgebra A contains all the atomic relations, the deductive closure of any problem over A can be computed in PTIME by resorting to the algorithm for A satisfiability [25].

For instance, let us consider the CA subalgebra of IA. This is tractable. CA relations can be represented as PA relations of the form (1) and (2). Checking the consistency of a constraint problem over CA can be done in quadratic time in the number of events by means of the algorithm for PA developed in [40]. Computing the deductive closure of a problem over CA can be done in cubic time in the number of events with the path consistency algorithm, which is a constraint propagation algorithm. Constraint propagation algorithms monotonically search the input problem for the minimum problem that satisfies a so-called local consistency property and has the same solutions as the input problem. In case of the path consistency algorithm, the local consistency property is that, for each triple of distinct events e_i, e_j and e_k, we have

$$C(i,j) = C(i,k)^T \bowtie C(k,j)$$

where $\bowtie$ is the composition operation over IA, and the operation T is so defined in terms of the inverse operation -1 over IA:

$$C(i,k)^T = \begin{cases} C(i,k) & \text{if } i \leq k \\ C(k,i)^{-1} & \text{otherwise.} \end{cases}$$

The algorithm then works by enforcing such a property by using the operations of intersection, inverse and composition of IA as shown in Table 2.1. This is the PC-2 algorithm of [24], that we took from [5].

V_0:=$\{(e_i, e_j, e_k) | i < j\}$;
V:=V_0;
while $V \neq \emptyset$ **do**
  choose $(e_i, e_j, e_k) \in V$;
  $C(i,j) = C(i,j) \cap \big(C(i,k)^T \bowtie C(k,j)\big)$;
  if $C(i,j)$ changed **then**
    add to V the subset of triples of V_0 that contain e_i and e_j;
  else
    remove (e_i, e_j, e_k) from V;
  fi
od

Table 2.1: A path consistency algorithm

In turn, this algorithm can be used to decide about the consistency of the maximal tractable subalgebra that contains CA, namely, the ORD-Horn subalgebra [25]. Computing the deductive closure of the ORD-Horn subalgebra can be done in time $O(n^5)$ by resorting to the path consistency algorithm, with n equal to the number of events.

However, neither the ORD-Horn subalgebra and, hence, nor CA allow for expressing disjointness, as in "before or after". Notice that Sp and Ep are the only maximal tractable subalgebras that allow for it [22]: Sp can viewed as the set of relations obtained by replacing each of the basic relations `meets`, `overlaps`, `during`, `finishes` and their inverses with their disjunction with `before`; Ep can viewed as the set of relations obtained by replacing each of the basic relations `meets`, `overlaps`, `during`, `starts` and their inverses with their disjunction with `before`.

2.3 The TERENCE Smart Game Design

According to [1], a game should specify (at least) the *instructions*, the *states* of the game, with the initial and terminal states, and the legal *actions* of the players. For specifying the data for the TERENCE smart games, we analysed the requirements for smart games resulting from contextual inquiries with c. 70 experts of poor comprehenders and educators as well as from field studies with c. 500 poor comprehenders. The results allowed us to classify and design smart games as follows.

Following experts of stimulation plans, in [4], the TERENCE smart games were classified into 3 main difficulty macro-levels as follows:

ML1. at the entry macro-level, *character* games, that is, either who the agent of a story event is (WHO), or what a character does in the story (WHAT);

ML2. at the intermediate macro-level, *time* games, for reasoning about temporal relations between events of the story, purely sequential (BEFORE-AFTER) or not (all others);

ML3. at the last macro-level, *causality* games, namely, concerning causal-temporal relations between events of the story, that is, the cause of a given event (CAUSE), the effect (EFFECT), or the cause-effect relations between two events (CAUSE-EFFECT). All games were specialised into *levels* arranged in a so-called linear layout, and designed as puzzle casual games [1].

They are presented in that order, so as to make learners first ponder about sparse events as in character games, then about their temporal relations, and finally about their causal-temporal relations. See Figure 2.3 for the taxonomy and the linear ordering of game levels (given by green arrows). Independently of that, the data for all levels are structured via the TERENCE game framework. See [4, 11] for a description of the framework and the design process.

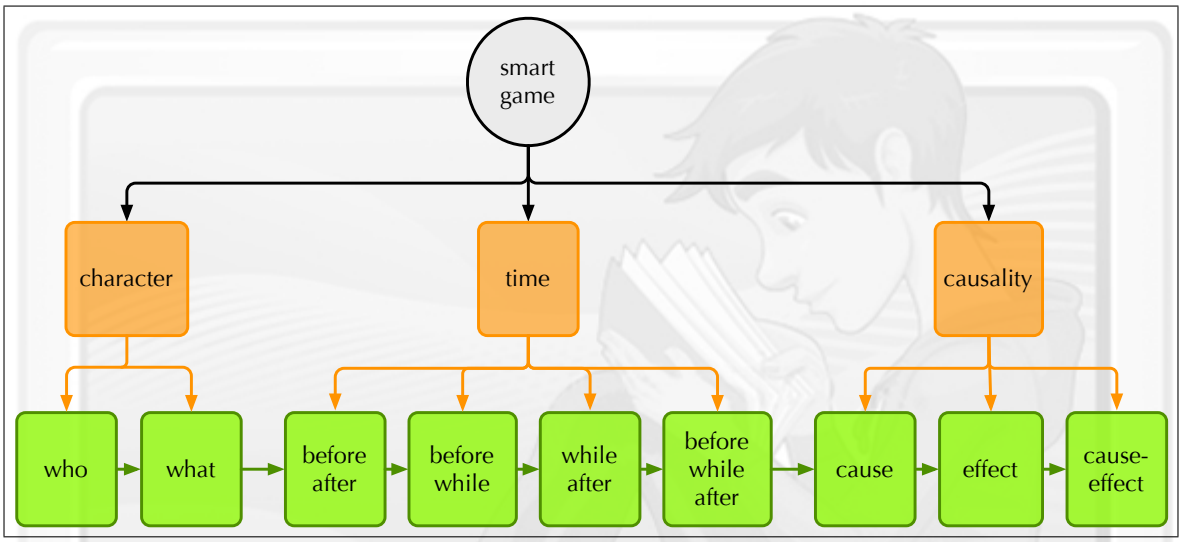


Figure 2.3: The taxonomy of game levels

Relevant fields of the framework for this paper are the following ones:

F1. the *question* of the game,

F2. a *central event* from the story,

F3. the *choices* for learner's actions, so that the availability of choices depend on the state the game is in, and

F4. which choices form a correct or wrong *solution*.

The fields are rendered with illustrations and textual components that vary according to the level of the game and the specific game instance, as explained below.

In case of WHO games, the question is related to a single central event that depends on the game instance, if the central event is "Aidan runs fast" then the question is "Who runs fast?". The choices are 3 characters and only one is the correct solution, namely, the agent of the central event named "Aidan". See the left top screenshot in Fig. 2.4.

In all the other levels, the question is related to the instructions for the game only, and is the same for all game instances. Choices are textual and visual descriptions of events from the story, and are placed at the bottom of the interface. In WHAT games, the central event is also the correct solution, and is placed at the bottom with the other choices that are wrong solutions. See the right top screenshot in Fig. 2.4. In time and causality games, the central event is at the centre of the interface. The choices are at the bottom and are story events that the learner has to correctly correlate with the central event. See the bottom screenshot in Fig. 2.4, a BEFORE-AFTER time game instance.

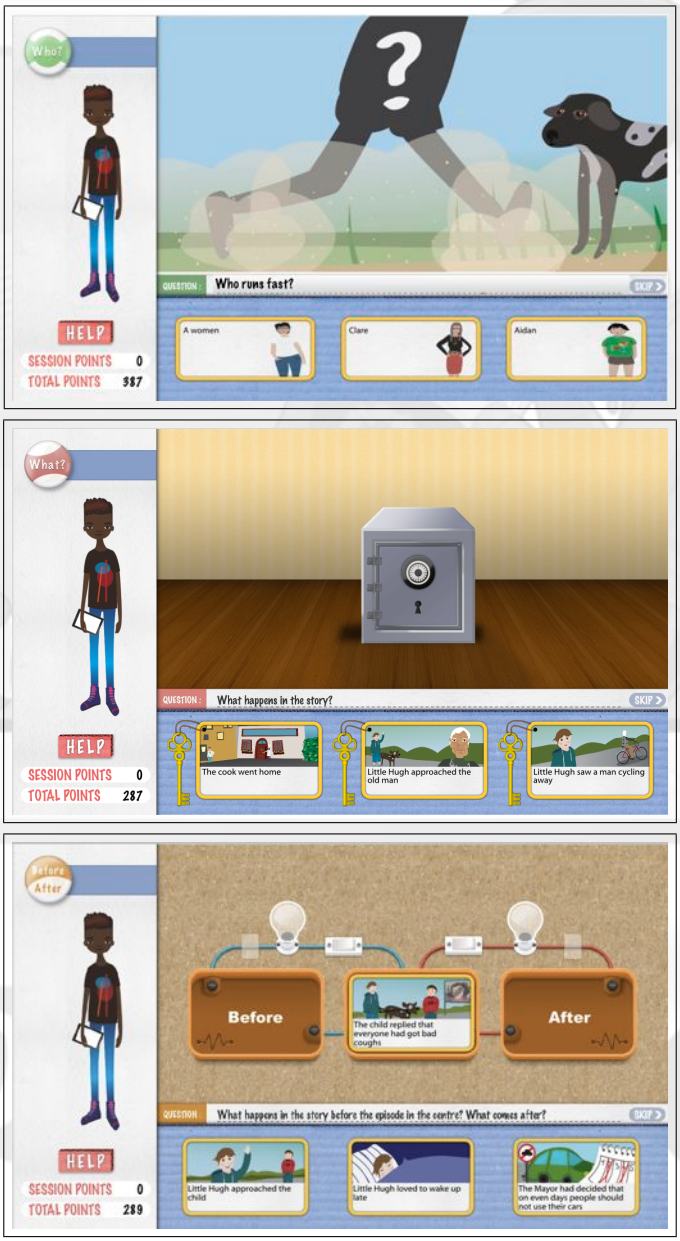


Figure 2.4: Screenshots of WHO, WHAT and BEFORE-AFTER game instances.

3 Enrichment of Annotations of Stories

3.1 Introduction

Preliminary to the generation of textual games is the reasoning about the annotations returned by WP3. The TERENCE reasoning module takes in input a story annotated by the NLP module from the annotated story repository of WP3, in Italian and English, and processes it for consistency checking and deduction, as explained below. The supporting architecture is explained in the end of this chapter.

3.2 Consistency checking

The static method `consistency` of class `GQR` serves to provide the TAF annotations of a story with a semantics over the real line and to check the consistency of the annotations with respect to the chosen semantics.

The method takes in input the TAF document annotated by the NLP module. The TAF document is converted into a TC problem file, with disjunctive Allen relations, see Section 2.2. The most critical task is to choose the 'right' semantics for `TLINKs`, that is, how to convert `TLINKs` into relations of a tractable subalgebra of the Allen interval algebra [2, 22] for the TERENCE stories.

The chosen mappings were two and agreed upon with the NLP partner as those returning inter-consistency among manual annotators, within pilot studies for English and Italian children stories. The hard mapping is reported in Table 3.1 and the relaxed on in Table 3.2. The relaxed mapping differs from the hard one in the interpretation of BEFORE, INCLUDES and their inverses. For both mappings the range is a subalgebra of the *continuous algebra* (CA), for which consistency checking and deduction take at worst cubic time in the number of events [19]. In fact, both consistency checking and deduction are done via the optimised path consistency algorithm of `GQR`, which runs in time cubic in the number of events.

The method parses the output, returning whether the document is consistent as a Boolean, and according to which semantics—the hard mapping or the relaxed one. If the story is consistent according to either one or the other semantics, deduction is invoked with the correct mapping—hard or relaxed. Else human interventions is invoked.

3.3 Deduction

The static method `deduction` of class `GQR` picks the semantics for which the annotations of a story are consistent and deduces further temporal relations via path consistency.

The static method `deduction` has in input the TAF document checked for consistency, with either the hard or relaxed mapping. The TAF document is converted into a TC file in the format of `GQR`, with the mapping for which the document is consistent. Then `GQR` is executed. The output of `GQR` is then parsed, and the deduced relations are added with `TLINKs` via the inverse mapping.

In case a relation r deduced by the reasoner corresponds to no one in the range of the adopted mapping for `TLINKs`, then r is approximated by the smallest (for inclusion) relation in the range of the mapping and that contains r, and a comment is added with the deduced relation for further processing—human or automated. The input TAF document is updated, and the updated TAF document is returned as XML file in the annotated story repository.

The story in the annotated story repository is referred to as the *enriched* story, used as input for the generation process described in Chapter 4 below.

`rel_types` for `TLINKs`	Allen relation(s)
BEFORE	`before`
AFTER	`before`$^{-1}$
OVERLAP	`equal` $\vee$ `during` $\vee$ `during`$^{-1}$ $\vee$ `overlaps` $\vee$ `overlaps`$^{-1}$ $\vee$ `starts` $\vee$ `starts`$^{-1}$ $\vee$ `finishes`$^{-1}$ $\vee$ `finishes`$^{-1}$
INCLUDES	`during`$^{-1}$
IS_INCLUDED	`during`
IDENTITY	`equal`
ALL	the disjunction of all Allen atomic relations

Table 3.1: Hard mapping of TLINKs into disjunctive Allen relations.

`rel_types` for `TLINKs`	Allen relation(s)
BEFORE	`before` $\vee$ `meets`
AFTER	`before`$^{-1}$ $\vee$ `meets`$^{-1}$
OVERLAP	`equal` $\vee$ `during` $\vee$ `during`$^{-1}$ $\vee$ `overlaps` $\vee$ `overlaps`$^{-1}$ $\vee$ `starts` $\vee$ `starts`$^{-1}$ $\vee$ `finishes`$^{-1}$ $\vee$ `finishes`$^{-1}$
INCLUDES	`during`$^{-1}$ $\vee$ `starts`$^{-1}$ $\vee$ `finishes`$^{-1}$ $\vee$ `equal`
IS_INCLUDED	`during` $\vee$ `starts` $\vee$ `finishes` $\vee$ `equal`
IDENTITY	`equal`
ALL	the disjunction of all Allen atomic relations

Table 3.2: Relaxed mapping of TLINKs into disjunctive Allen relations.

3.4 Architecture

The architecture of the reasoning module is made up of: (1) a REST service, that contains two main operations: consistency checking and deduction; (2) a Java library `tml2gqr` that is used by the service for implementing the operations, but can be also embedded in other applications for performances' reasons; (3) the GQR software [16], invoked by the library for performing the above operations. See Figure 3.1 for an overview.

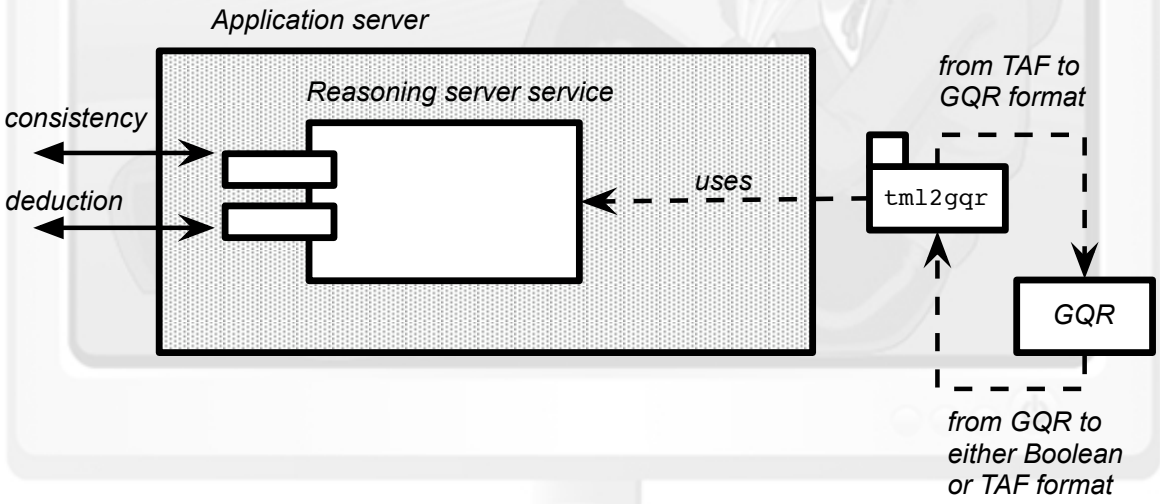


Figure 3.1: Architecture

4 Generation of Textual Games

4.1 Introduction

The reasoning module also takes care of generating textual games—that is, specific textual components of instances of smart games in XML, that vary from textual game to textual game. An example will suffice for all: correct and wrong solutions for a game instance.

For the generation of textual games, the reasoner works with input the enriched XML stories, returned enriched by the reasoner as explained in Chapter 3. Then it automatically generates textual game instances, for each level (see Section 2.3), according to specific selection heuristics. Finally, textual games, generated automatically with a click, are manually evaluated and revised. The entire process is described in [3, 11]. In this chapter, we delve into the automated generation process of textual games.

For the automated generation, the reasoner performs the following main steps:

Step (1): the reasoner generates *basic* textual games, namely, textual games made only of their identifier, level, central event and solutions, wrong and correct;

Step (2): the reasoner ranks such basic textual games according to specific heuristics;

Step (3): the reasoner selects the basic textual games at the top of the ranking, and prunes the others;

Step (4): for each basic textual game, for each event in it, the reasoner invokes the NLG module of the pertinent language that generates the necessary textual information for the event, as outlined in Chapter 5 below;

Step (5): finally, for each basic textual game, the reasoner places the generated text in the appropriate data structure, and returns the related textual game in XML format.

See Fig. 4.1 for the overall workflow.

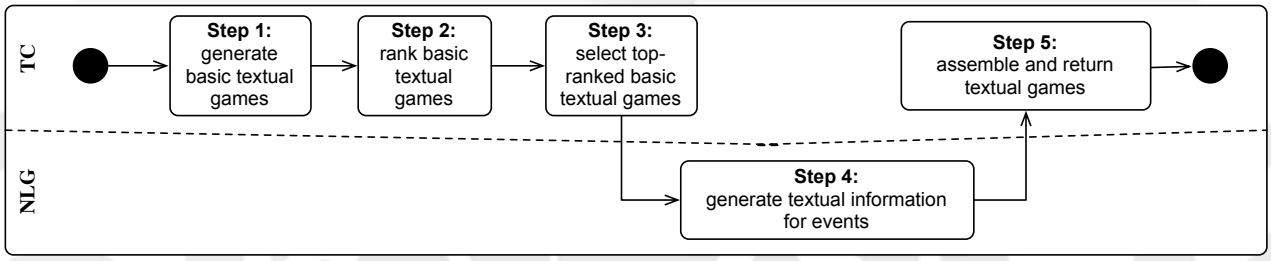


Figure 4.1: The generation workflow

This chapter describes the ranking strategy and the first step in more details. The natural language generation algorithms for the second step are in Chapter 5.

4.2 Ranking and Selection Strategies

We now focus on Step 2 of Figure 4.1, that is, on the strategy for the selection of the required number of textual games for the TERENCE learners, according to the TERENCE stimulation plan [4]. In the first releases of D4.2 and D4.3, the selection was randomly done. In the second and final releases, we introduced a ranking

steps with heuristics for performing a better selection of games. In this manner, the generation algorithms operate with a global view of the global generation process right after the first step of the process, when the total number of basic games generated is higher (even greatly) with respect to the number of required textual games. In the following, we first outline the overall ranking strategy and then we sketch the heuristics that allow for the selection of basic textual games.

4.2.1 Outline of the Ranking and Selection Algorithms

In Step 2 of the process, central events are ranked according to how frequently they occur in time and causality basic textual games, from the most frequent to the least frequent. Then all generated basic textual games are given a point for each satisfied heuristics, described in Subsection 4.2.2 below.

When such a ranking step is over, we move to Step 3 for the selection: if we need to generate n textual games, we select the n games with the most frequent central event and highest number of satisfied heuristics.

The implemented heuristics are detailed in the remainder of this section per game level.

4.2.2 Heuristics for character games

Heuristics for one of the wrong choices of WHO games, so as to make them more plausible:

- if the correct choice has PROTAGONIST as value for story_role, then one of the wrong choices should have PROTAGONIST or SECONDARY as well as values for story_role;
- a character that is a wrong choice should have the same value for entity_type as the character that is a correct choice; e.g., if the correct choice is an ANIMAL, then one of the wrong choices should be an ANIMAL as well.

Heuristics for the correct choices of WHAT games, so as to make it plausible: a participant of the event of the correct choice (that is from the current story) has story_role equal to MAIN.

Heuristics for one of the wrong choices of WHAT games, so as to make it plausible: the event of a wrong choice (from an unread story of the same book, or from the first read story if the current story is the last) shares a participant with the event of the correct choice.

4.2.3 Heuristics for time games

Heuristics for the wrong choices of BEFORE-AFTER games, so as to make them plausible: choose as wrong choice an event that is in a TLINK relation with value INCLUDED_IN with the central event.

Heuristics for the wrong choices of BEFORE-WHILE games, so as to make them plausible: choose as wrong choice an event that is in a TLINK relation with value AFTER with the central event.

Heuristics for the wrong choices of WHILE-AFTER games, so as to make them more plausible: choose as wrong choice an event that is in a TLINK relation with value BEFORE with the central event.

4.2.4 Heuristics for causality games

Heuristics for the wrong choices of CAUSALITY games, so as to make them plausible: choose as wrong choice an event that shares a participant with the central event or the correct choice.

4.3 Algorithms for the Generation of Smart Games

In the following, we explain how choices, correct and wrong solutions are generated and the NLG modules are invoked, according to the macro-level of games we deal with. We also give the pseudo-code of each generation algorithm.

4.3.1 Character games

In order to create a WHO game instance for an event, we exploit the existence or not of a chain that, starting from the event marked with `Event`, continues to the participant related via `hasParticipant` and marked with `Entity Mention`. This is then resolved by correlating it via `refersTo` to its `Entity`. Fig. 4.2 depicts the case of two entities, i.e., Ben and Kate, mentioned in the story as “The boy”, and “The girl”, respectively, and their participation in two events, i.e., “swim” and “run”.

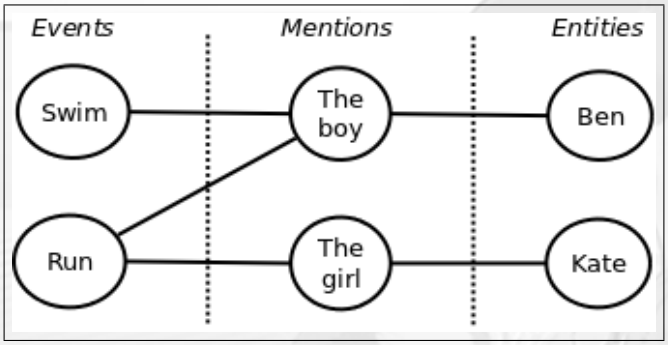


Figure 4.2: TERENCE TimeML tags for a WHO smart game instance.

For instance, to generate the WHO game for the central event “swim”, the module exploits the links that correlate

- “swim” to the respective mentions, then to the actual entities (in the example, Ben), for the correct choice,
- other entities that are not mentioned as participants in the “swim” event as wrong (e.g., Kate).

A similar process takes place with WHAT character games. Given an event in the story, this is taken as the correct choice, whereas the wrong choices are from either a different story, or from the same story but using a different entity as event participant. In the case depicted in Fig. 4.2, the correct choice would be “Ben swims”, and a wrong choice would be “Kate swims”.

The following is the pseudo-code of the core algorithm for the generation of a textual instance of a who game and of a what game.

THE WHO-GAME ALGORITHM
whogame(TEXT_XML enriched j, TEXT_XML enriched k, relevant event e_i) % choose correct and wrong solutions select an agent of e_i in TEXT_XML enriched j; read entity description of character; print entity description of character as correct choice; select a character that is not a participant of e_i in TEXT_XML enriched j; if none then select a character that is not a participant of e_i in TEXT_XML enriched k; if correct choice is PROTAGONIST or SECONDARY then wrong must be PROTAGONIST or SECONDARY read entity description of character; print entity description of character as wrong choice 1; % select a character that is not a participant of e_i in TEXT_XML enriched j; if none then select a character that is not a participant of e_i in TEXT_XML enriched k; if correct choice is PROTAGONIST or SECONDARY then wrong must be PROTAGONIST or SECONDARY read entity description of character; print entity description of character as wrong choice 2; % % invoke the NLG module ask the NLG module for who question of e_i in TEXT_XML enriched j; print the who question;

THE WHAT-GAME ALGORITHM
whatgame(TEXT_XML enriched j, TEXT_XML enriched k, relevant event e_i) % Choose correct solutions select event e_i; print event as correct choice 1; % choose wrong solution select an event e_j from TEXT_XML enriched k; print event as wrong choice 1; % other wrong solution select an event e_k from TEXT_XML enriched k; print event as wrong choice 2; % % invoke the NLG module ask the NLG module for the game events of e_j and e_k in TEXT_XML enriched k; print the game events;

4.3.2 Time games

In order to create a time game with a given central event, we exploit the `Event` and `TLINK` tags, as well as the `local` and `rel_type` attributes. Fig. 4.3 shows a portion of the temporal structure of a story, where the event "swim" occurs before "run", "jump" occurs at the same time as "run", and "rest" occurs after "run". Furthermore, two TLINKs are not local and were deduced by the TC module in the story annotation process, i.e., that "swim" occurs before "rest".

For instance, in order to create a BEFORE-AFTER-game instance difficult for a poor comprehender, we can:

- exploit the `TLINK` that has a `rel_type` equal either to "before" or "after",
- consider relations with `local` set to false (i.e., jump-rest, swim-rest).

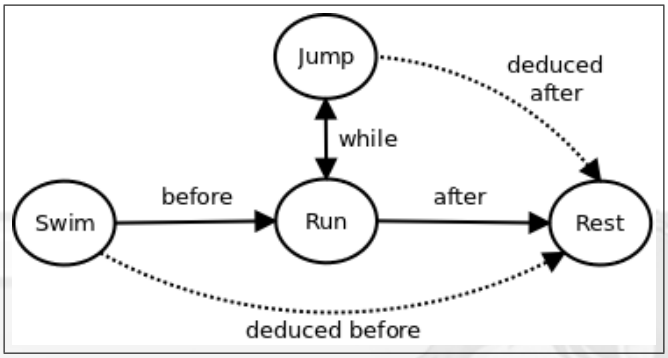


Figure 4.3: TERENCE TimeML tags for time smart game instances.

The following is the pseudo-code of the core algorithm for the generation of a textual instance of a before-after game. The pseudo-code for the before-while, while-after and before-while-after games is similar.

THE BEFORE-AFTER-GAME ALGORITHM

```
bagame(TEXT_XML enriched j, TEXT_XML enriched k, relevant event e_i)
% choose events e_i, e_j and e_k so that e_i is BEFORE e_j and e_k is AFTER e_i
select 3 events e_i, e_j and e_k from TEXT_XML enriched j where
e_i is relevant or e_j is relevant or e_k is relevant;
e_i is target of TLINK x with value BEFORE and of TLINK y with value AFTER;
e_j is source of TLINK x with value BEFORE;
e_k is source of TLINK y with value AFTER;
print e_i as central event;
% correct BEFORE and AFTER
print e_j as BEFORE event;
print e_k as AFTER event;
% wrong event, working as distractor: a WRONG CHOICE should be an event
% from the current story that is in the relation INCLUDES or
% INCLUDED_IN with FE.
if relation(event e_l, central_event) = 'INCLUDES' or
   relation(event e_l, central_event) = 'INCLUDED IN'
   select event e_l from TEXT_XML enriched k
   print e_l as WRONG event;
else
   % Select a random event form other story
   select random event e_l from TEXT_XML enriched other
endif
% invoke the NLG module
ask the NLG module for the game events of e_i, e_j, e_k in TEXT_XML enriched k
   and of e_l in TEXT_XML enriched l;
print the game events;
```

4.3.3 Causality games

In order to create causality games with a given central event, we exploit `CLINKs` and the `reverse` attribute, mainly. Fig. 4.4 depicts causal relations in a story where a race was lost because the participant did not sleep enough, and that caused the participant to become sad.

For instance, to create a CAUSE-game, we can exploit the `CLINKs` stating that the cause for "The boy is sad" is that "He lost the race", and properly swap the cause and the effect, in case the `reverse` attribute is set to true ("The boy lost the race" because "He did not sleep").

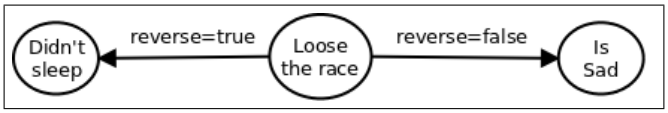


Figure 4.4: TERENCE TimeML tags for causality smart game instances.

The following is the pseudo-code of the core algorithm for the generation of a textual instance of a causality game. The pseudo-code for the effect and cause-effect games is similar.

THE CAUSE-GAME ALGORITHM

```
causegame(TEXT_XML enriched j, TEXT_XML enriched k, relevant event e_i)
% choose events e_i, e_j, e_k and e_l so that e_j CAUSES e_i, and
% it is not the case that e_k CAUSES e_i or e_l CAUSES e_i
select 2 events e_i and e_j from TEXT_XML enriched j where
% correct CAUSE and central event
   e_i is relevant or e_j is relevant;
   e_i is target of a CLINK x with value CAUSES;
   e_j is source of a CLINK x with value CAUSES;
print e_i as central event;
print e_j as correct event;
% wrong CAUSE
select 2 events e_k and e_l from TEXT_XML enriched j where
   for each CLINK y where e_i is target do
      check that e_k is NOT source of CLINK y with value CAUSES;
      check that e_l is NOT source of CLINK y with value CAUSES;
print e_k as wrong event;
print e_l as wrong event;
% invoke the NLG module
ask the NLG module for the game events of e_i, e_j, e_k, e_l in TEXT_XML enriched k;
print the game events;
```

5 Generation of Game Events and Who Questions

5.1 Introduction

In order to complete the generation work of the reasoner, language-dependent components of textual games are generated via NLG technologies. These generate who-questions for WHO-games, as well as game events for all other games, that is, simplified sentences describing story events. Game events and who-questions are in general simple sentences that should be easy to read for all the TERENCE learners as prescribed by the TERENCE stimulation plan, which requires smart games to mainly rely on visuo-perceptual skills [4].

Therefore, the sentence simplification and question generation workflow was inspired by the following linguistic and cognitive principles, in order to make reading as simple as possible for all TERENCE learners.

- **Extract only factual events**: The goal of simplification is to obtain a list of simple statements containing the events described in a story. This should support children in understanding the sequence of events that actually took place in the story. For this reason, we focused on *factual* events, i.e., those that have a clear time span within the story and really happened in the narrative. This led us, for instance, to discard as candidate events verbs in the conditional mood and in the future tense.
- **Make information explicit**: Each simplified statement should be understandable, self-contained, with no implicit information. This is particularly crucial in Italian, because it is a pro-drop language in the case of subject pronouns, and the sentences of a story in isolation may miss the information on who performed the action. Therefore, we introduced an anaphora resolution module aimed at replacing personal pronouns with their overt antecedent.
- **Retain all and only necessary information**: Each simplified statement should describe an event and its participants in a well-formed sentence. Each piece of information reported should be necessary to make the event understandable, avoiding all superfluous elements that could hamper the reading of poor comprehenders. At syntactic level, this corresponds to eliminating all adjuncts, which however is not a trivial task, since existing parsers cannot distinguish between arguments and adjuncts with good precision. We devised a simplification strategy to cope also with possible parsing errors (for details, see Section 5.3.3).
- **Report events in the present tense**: The output of the simplification process should be a list of self-contained statements, which can be used to generate questions on the events, or be included in the games generated from the story. In order to use them in isolation in any of such contexts, the event has been reported in the present tense.

The NLG modules implemented for English and Italian show some differences due to the availability of different NLG technologies. Nevertheless, the general principles on which the simplification modules are based are common between the two languages, as described above. Another difference lies in the simplification workflow: while the English module simplifies only the sentence containing a given event, the Italian one first simplifies all factual events in the whole story, and then returns only the simplified statement containing the given event. The second choice was motivated by the fact that the simplification module is intended to become a standalone tool [6].

5.2 English

5.2.1 Game Sentence and Who-question Generation Framework

The English NLG module regenerates the game sentences and WHO-questions for the TERENCE smart games in the following sequence.

- The information of the events and entities are extracted.
- We parse the sentence to a syntactic tree (we use the Stanford parser[1])
- We locate the relevant context of the event.
- We change the event verb to the right form (in the present tense) suited with the subject of the event.
- We form up the game sentence with Subject (use its co-reference), verb and its VP extracted from the syntactic tree.
- We form up the WHO-question by simply replacing Subject of the game sentence to “Who”

5.2.2 Approach description

5.2.2.1 Information Extraction

We extract morphology, semantic roles and dependency information related to events and entities for sentence regeneration tasks. To do this, we leverage on the results of the annotation module of [9] for extracting the following information:
Events: predicate, polarity, aspect and id. We name the event that is annotated by the NLP module *annotated event.*
Entities: participant role, id, referred event that the subject is involved.
Co-references: source and target entities in the story
Tokens: the original sentence of the event mentioned in the story.
For example,
a. Events

```
 <EVENT comment="" pred="to smile" polarity="POS" aspect="PROGRESSIVE" id="968">
  <token_anchor id="591"/>
</EVENT>
```

b. Participants: subjects and objects

```
 <HAS_PARTICIPANT ita_dep="undef" eng_dep="NSUBJ" id="1075">
        <source id="948"/>
        <target id="876"/>
</HAS_PARTICIPANT>
```

c. Co-references

```
 <REFERS_TO id="1078">
        <source id="872"/>
        <target id="1018"/>
</REFERS_TO>
```

d.Tokens

[1] http://nlp.stanford.edu/software/lex-parser.shtml

```
<token number="0" id="1" sentence="0">Ben</token>
<token number="1" id="2" sentence="0">and</token>
<token number="2" id="3" sentence="0">Sophie</token>
<token number="3" id="4" sentence="0">were</token>
<token number="4" id="5" sentence="0">twins</token>
<token number="5" id="6" sentence="0">.</token>
```

e. Entities

```
<ENTITY ent_desc="Sophie" ent_type="PERSON" story_role="PROTAGONIST" id="1018"/>
<ENTITY_MENTION comment="" reference_type="SPC" syntactic_type="NAM" id="899">
            <token_anchor id="479"/>
</ENTITY_MENTION>
```

Enrich event information

One issue that sometime occured is that the EVENT may miss the attribute "pred=" inside the <event> tag, for example,

<event id="1522" morphology="NONE" polarity="POS" modality="NONE" pos="ADJECTIVE">grateful</event>

the mentioning "grateful" is annotated but in the attribute list, there is no "pred=" attribute because errors happened in automatic event recognition. One straightforward solution is to use the event mentioning directly, however, it faces grammatical issue on agreement among sentence elements. In the example above, "grateful" is an adjective, not a verb, hence the system should not use it as the predicate when generating game event sentence. To overcome this issue, we look into the original sentence and locate the token that possibly becomes the event predicate. We apply state of the art syntactic parser [2] to find the verb of the parsed sub-tree that contains the event mentioning and then consider it as the predicate of the event.

5.2.2.2 Determine the relevant context of the event

Determining the relevant context of the event aims at locating the context of this event from the original sentence. We notice that sentences of our children stories are well-formed in the sense of sentence structures. That suggests us that we can apply heuristic algorithms working on the syntactic parsed tree of the original sentence to locate the relevant information.

To do so, we parse the original sentence into a syntactic tree. Next, we find the sub verb-phrase tree (VP) whose head is the *Predicate* of the considered event. Then we take all leaves of this sub-tree and apply some heuristic rules to locate the context of the event. Our heuristic rules are manually created as follows:

– *If there exists "," or "and" inside the VP:*

  +rule #1: if there is a S tree with subject (S (NP, VP)) after the comma or *'and'* , which means there is another clause inside the VP tree which can form a separate sentence, we remove that S tree and stop right before the commas or *'and'*.

Event	**Original Sentence**	**Relevant context**
paddle	Ben was paddling fast, still he didn't make it to the end wall of the pool before Luke	Ben was paddling fast

Table 5.1: An example of relevant context of an event

  +rule #2: If there is no S tree with subject after the comma or "and" neither an annotated event, we take all the VP tree, since it could be the important information.

Event	Original Sentence	Relevant context
visit	Ben and Sophie visited the new Sport Complex in town with Mum **and** her friend from work	Ben and Sophie visited the new Sport Complex in town with Mum **and** her friend from work

Table 5.2: An example of relevant context of an event

+rule #3: If there does not exist a S tree after the comma or *'and'*, but exists an annotated event in side the VP tree, we remove this sub VP tree of this annotated event.

Event	Original Sentence	Relevant context
count	They count one two three, $diving_{annotated\_event}$ into the cool water of the swimming pool	They count one two three

Table 5.3: An example of relevant context of an event

– *If there exists a sentence connective other than "and"*

for example, "when, while, until, because, since, but" inside the VP tree:

+rule #4: If it is followed by another conjunction having subject and verb inside the VP, we ignore the conjunction of this event.

Event	Original Sentence	Relevant context
think	He was thinking hard while the children looked at him expectantly	He was thinking hard

Table 5.4: An example of relevant context of an event

+rule #5: If it is **not** followed by another conjunction without subject and subject inside the VP, we do not ignore them.

– *If there is neither no connectives nor "and", comma:*

+ rule #6: If there exists another annotated event *inside* our tree which *has subject* explicitly , we look for its S tree of this event and exclude it from the context.

+ rule #7: If there exists another annotated event *inside* our tree which *does not have subject* explicitly , we take the whole VP tree to the content of the event.

5.2.2.3 Form the subject and resolve co-references

We extract the list of *Subject* participants from the *Event* data structure. We form the subject of the "will be generated" game event sentence in the "make sense" form. We apply some following rules for subject formation:

[2]Stanford Parser: nlp.stanford.edu

Event	Original Sentence	Relevant context
think	He thought of the car while playing	He thought of the car while playing

Table 5.5: An example of relevant context of an event

Event	Original Sentence	Relevant context
get	Everyone gets excited except for Mum who is sulking in her bedroom	Everyone gets excited except for Mum

Table 5.6: An example of relevant context of an event

+ rule #8: if the sentence has only one subject participant, the subject is the description of this participant itself.

+ rule #9: if the number of Subject participants is two, we form it $Participant_1$ and $Participant_2$, for example “Ben and Luke”.

+ rule #10: if the number of Subject participants is more than two, we make them $Participant_1$, $Participant_2$, *and* $Participant_3$, for instance, “Ben, Sophie and Luke”.

In addition, the entities, including the subject participants as well as other mentioned entities, needs co-reference resolution. However, not every cases we resolve them. We follow some manually created rules:

+ rule #11: if the entity is mentioned as a reflective pronoun, such as “himself, herself”, do not resolve the entity. For example: “Annabel made apple jam *herself*”, “herself” is an entity mention but it should not be resolved as “Annabel” in the generated sentence.

+ rule #12: if the entity is mentioned twice in the sentence, we just resolve it at the first time it occurs and keep it as it is at the second time. For example, “Ben was paddling as fast as he could”, we do not resolve the mention “he” to “Ben” in order to make the generated sentence more fluent.

+ rule #13: we do not resolve the possessive pronouns, such as “her, his, its, yours, theirs”. For example: “Annabel decides to make some nice apple jam for her new friends.”, we do not resolve the mention “her” here.

To do so, we start from the information we extracted from the annotation module to locate the position of the participants on our extracted context tree. We replace the text of the nodes, of which each is currently describing un-resolved description of the entities, with the description of their co-references.

5.2.2.4 Form the complete sentence and Who question

We join the subject, verbs and the context of the event from our context tree to form the game event sentence. For generating the WHO question, we simply replace the subject by “Who”.

5.2.2.5 Post processing

Since there exists un-annotated verbs that belong to a phrase supporting the meaning of the event in the generated sentence. These verbs are not in the present (continuous) form yet. That requires us to change these verbs to the present (continuous) form.

For example:

Event	Original Sentence	Relevant context
see	Ben saw his sister coming down the steps	Ben saw his sister coming down the steps

Table 5.7: An example of relevant context of an event

Original Sentence	Transform to present tense
Ben saw his sister coming down the steps	Ben sees his sister coming down the steps
He was running as fast as he could	He is running as fast as he can

Table 5.8: Example of transforming the verb to the present tense

Our target is to transform all verbs of the generated sentence to the present tense. To do so, we use PoS tagging approach to locate all verbs of the generated sentence, basing on their POS tagging information.

We use the verb dictionary of verb morphology published by NodeBox[3] to get the present or present continuous form of the verbs. The verb dictionary is simplified into the following form:
<**I**nfinitive **S**ingleTense **G**erund>

We apply some following rules for verb changing:

+ rule #14: if the event's aspect is "progressive" and the subject is "plural", we make it' "are" + verb-ing

+ rule #15: if the event's aspect is "progressive" and the subject is "single", we make it "am/is" + verb-ing depends on the subject is "I" or not.

+ rule #16: if the event's aspect is **Not** "progressive", we change the verb to the present tense.

5.2.2.6 Evaluation

We developed a corpus for testing our sentence generation modules. The corpus includes 9 stories designed for children in the Terence project and the gold standard (ground truth) data, which is consist of manually created 287 game sentences corresponding to those events extracted from above 9 stories. We used the BLEU[26] and TER[37] metric to measure the quality of game event and who question generation task by comparing the generated game events with the sentences in the gold standard data.

In brief, BLEU and TER are typical methods that are widely used for evaluate the quality of automatic machine translations by measuring how similar the generated sentence to the gold standard sentence. BLUE relies on the number of n-gram overlaps between them and TER computes the number of modifies such as delete/shift to transform a generated sentence to the gold standard sentence. BLUE score equals to 1 or TER score equals to 0 means the generated sentence is perfectly matched to the gold standard sentence.

The result is shown in the table 5.9.

ID	$Events$	TER	$BLEU$	ID	$Events$	TER	$BLEU$
1	24	0.14	0.85	3	26	0.31	0.57
2	27	0.46	0.53	4	29	0.39	0.66
5	41	0.36	0.62	6	25	0.34	0.59
7	39	0.49	0.53	8	42	0.31	0.66
9	34	0.30	0.71	**Average**	**32**	**0.34**	**0.63**

Table 5.9: Experiment results on 287 events from our corpus of 9 stories

The system outputs overall BLEU score is 63 and TER is 0.34.

5.3 Italian

The Italian NLG module is composed by several processors, displayed in Fig. 5.1.

[3]http://nodebox.net/code/index.php/Linguistics

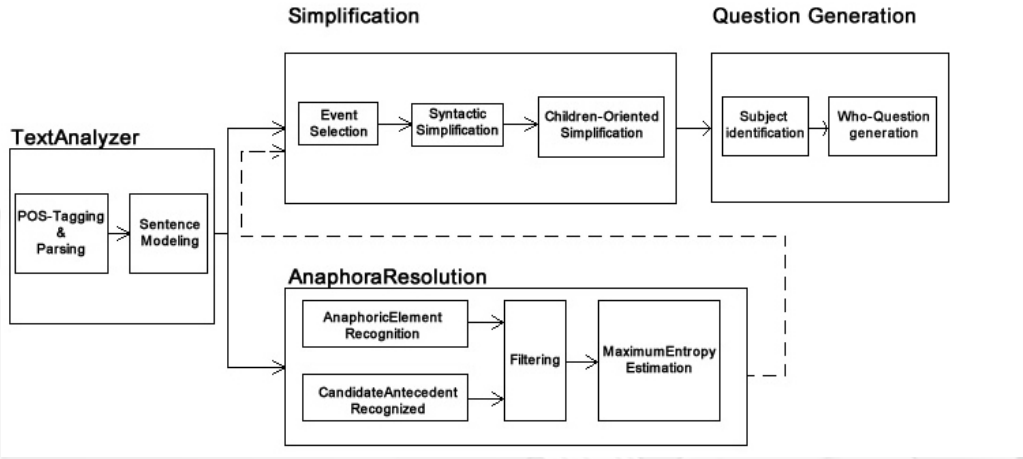


Figure 5.1: Sentence simplification and question generation workflow

The Text Analyzer component performs the linguistic pre-processing needed for the two following steps. Then, the Anaphora Resolution component identifies the antecedent of pronouns / subjectless verbs. In the next step, the Simplification component performs sentence simplification. It can take in input the text with resolved anaphoras, as produced by the previous module, or operate directly on the pre-processed text. Finally, questions are generated from the simplified statements. Details on the different modules composing the overall architecture are reported in the following subsections.

5.3.1 Pre-processing with the Text Analyzer component

Given an input story, the first step towards its simplification consists in applying a set of NLP tools to extract basic lexical, morphological, syntactic and semantic information. We use TextPro [27], a freely available NLP suite for Italian, to tokenize and lemmatize the text, as well as to provide morphological information on each token. Besides, we run also the in-built named entity recognizer, whose information will be used in the anaphora resolution step.

Next, the Italian version of the Malt parser [23] is used to obtain a syntactic analysis of each sentence, with the tokens being pairwise connected by labeled dependencies[4]. While past works have mainly relied on constituency information to perform syntactic simplification of sentences (see e.g. [7] and [8]), we implement a simplification strategy based on argument/adjunct information, and therefore we integrate a dependency parser in the processing pipeline. Our choice is corroborated by recent findings on dependency-based simplification ([34], [35]). This is another difference with respect to the English simplification module, which relies primarily on a constituency parser.

5.3.2 Anaphora resolution

Anaphora resolution is crucial because each simplified sentence must be understandable in isolation, with no relevant information left implicit. This is particularly relevant for Italian texts, in which zero-anaphora is very frequent. Therefore, the simplification module includes an anaphora resolution component aimed at identifying the antecedent of (i) personal pronouns and (ii) null subjects (zero anaphora). Other anaphoric elements may be present in a text, for instance relative pronouns or possessives. However, we focus primarily on these two types of anaphora resolution, which are particularly relevant and frequent in Italian, while we leave the extension and refinement of this module to future work.

[4]The parser achieved 86.5% accuracy on relation labeling and 90.96% accuracy on head finding in the parsing task at Evalita 2009 evaluation campaign, see `http://www.evalita.it/2009/tasks/parsing`.

Since no anaphora resolution system is freely available for Italian, we developed it by combining a rule-based selection of anaphoric elements and possible antecedents with supervised classification.

Anaphora resolution is performed in four steps, as displayed in Fig. 1: first, anaphoric elements, which need to be resolved by finding an antecedent, are recognized. We basically extract all personal pronouns and verbs without subject. We further exclude from the verbs to resolve some typical verbs which do not have a subject in Italian, for instance meteorological verbs such as 'to rain', 'to snow', etc. Then, the candidate antecedents are collected by considering all nominal phrases that are directly depending on a verbal argument and that appear in a window up to three sentences before the anaphoric element to resolve. Then, we filter out candidate antecedents whose person, noun (and possibly gender) do not match with those of the anaphoric element. Finally, we classify each candidate pair including a possible antecedent and an anaphoric element as co-referring or not.

The features considered take into account lexical and morphological information, the distance between the anaphoric element and the antecedent, as well as the dependency label of candidate antecedents in the dependency tree. The complete list is reported in Table 5.10. All linguistic information needed to extract the features relies on the output of the pre-processing phase.

Feature class	Features
Grammatical role of candidate antecedent	Dependency label (e.g. SUBJ, OBJ, etc.)
Grammatical role of anaphoric element (if pronoun)	Dependency label (e.g. SUBJ, OBJ, etc.)
Position	Candidate antecedent is at sentence begin
Position	Anaphoric element is at sentence begin
Position	Anaphoric element and antecedent in the same sentence
Position	Token-based distance between element and antecedent
Dependency	Anaphoric element and antecedent depend on same verb
PoS of anaphoric element	Verb or Pronoun
Entity	Entity type of antecedent (PER, LOC, GEO, OTHER)

Table 5.10: Feature list for anaphora resolution

We train the maximum entropy classifier implemented in OpenNLP[5] with the data made available at Evalita 2011 evaluation campaign[6]. A first evaluation of the anaphora resolution module is reported in Section 5.3.5.2.

5.3.3 Simplified statement of factual events

In the following step of the workflow, the proper simplification process is carried out exploiting the linguistic information extracted in the previous steps. In particular, three subtasks are performed:

- **Identification and selection of factual events**: Factual events correspond to actions that took place in the story. In order to identify them, we select verbs based on their tense and mood information. Specifically, we discard all verbs in conditional mood, because they typically describe potential actions that may not happen. We also discard events expressed in the future tense, because the reader cannot assess if they will actually take place.
- For each factual event, **identification and selection of the mandatory arguments**: for this step, a set of rules has been implemented according to the verb valence. For transitive verbs (specified in a separate list), we discard arguments that are neither a subject nor a direct object. For other verb types, we identify the subject and the first modifier as mandatory, in case the latter is present. This rule has been introduced in order to alleviate an issue with the parser in distinguishing between arguments and adjuncts: this is a very difficult task in Italian, and the parser performance on this specific distinction is quite poor. Therefore, we implement a general rule that does not distinguish between the two.

[5] http://opennlp.apache.org/
[6] http://www.evalita.it/2011

– **Event reformulation**: after the sentence has been simplified, the verb needs to be expressed in the present indicative tense. This task may seem trivial, nevertheless some issues need to be taken into account, for instance the distinction between a passive verb at present tense (e.g. 'sono visto', transl. 'I am seen') and an intransitive verb at the past tense (e.g. 'sono andato', transl. 'I have gone'), because in Italian they are both composed by the auxiliary 'essere' (to be) followed by a past participle. Again, we implement some rules to alleviate TextPro errors in distinguishing between active and passive verbs, and then we conjugate the verb accordingly. The conjugation is performed by FSTAN, a generator of morphological forms included in TextPro.

As an example, we show the simplification process of the following sentence:

(1) Ernesta stava mangiando la torta con i suoi amici.

Ernesta was eating a cake with her friends.

The parser output is displayed in Fig. 5.2: after the main event 'mangiando' ('eating') has been extracted, the module looks for its arguments labeled by the dependency parser. In this example, the verb is the head of a SUBJ, a DOBJ and a RMOD. Since the simplification rule implemented for transitive verbs retains only the subject and the direct object, the RMOD subtree is first removed. Then, 'stava mangiando' ('was eating') is replaced by the present indicative form 'mangia' ('eats') . The simplified sentence is displayed in Fig. 5.3.

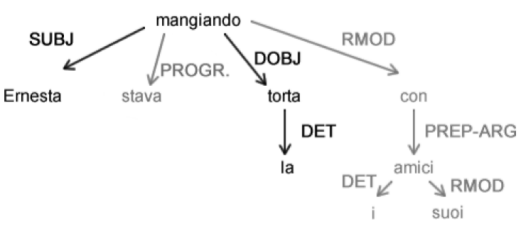


Figure 5.2: Sentence before simplification

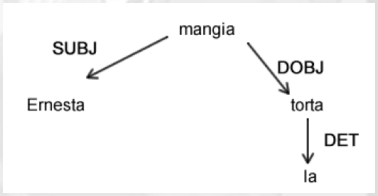


Figure 5.3: Simplified sentence: *'Ernesta eats the cake'.*

5.3.4 Question generation

After the simplified statement has been generated, it is given in input to a question generation system that automatically asks who performed the event (*Who-Question*). The task for Italian may be easier than for English, because Italian sentences follow a Subject Verb Object (SVO) structure also in interrogative clauses. In order to generate questions on the subject of a simple declarative clause, the NP that starts the sentence and precedes the verb can be easily identified as the subject. Then its corresponding entity type has to be found (for instance by using a NER and WordNet), and the corresponding question word ('who' or 'what') can be joined to the following verb. The process is displayed in Example (2), with the declarative sentence being the simplified clause generated by our system. This process is in line with [20], who transform sentences into questions by first expanding the source text into a set of derived declarative clauses.

(2) Ernesta mangia la torta. → Chi mangia la torta?

Ernesta eats the cake → Who eats the cake?

5.3.5 Experimental Setup and Evaluation

We perform evaluation on three different tasks, namely (i) event extraction, (ii) anaphora resolution and (iii) sentence simplification. Since the implementation of the modules is still in progress, these numbers are preliminary. At the moment, an evaluation of the question generation component is also missing.

5.3.5.1 Evaluation of event extraction

The first subtask we evaluate is event extraction: since we want to extract only factual events, describing what really happened in a story, we first assess whether the correct verbs are selected. This evaluation is performed only on Italian stories, because the module for this language generates all simplified statements for a whole story. This means that factual events must be automatically selected. The English component, instead, only simplifies statements containing the event given for game generation.

We create a gold standard composed of 6 children stories, asking annotators to select only the sentences whose main verb describes a factual event. Specifically, they were asked to select an event only if they would be able to put it in a timeline and connect it temporally to the other events in the story (an intuitive way to select only factual events). The total number of events in the gold data set is 320. Then, we compare the events recognized as factual by our module with those in the gold data set and we compute standard precision, recall and F1. The system achieves **P 0.88**, **R 0.79** and **F1 0.83** (macro average).

Although the system performance is generally good, there are few sources of errors which negatively affect event extraction. The first one is the wrong PoS label assigned to the verb during the pre-processing step (Section 5.3.1): the PoS-tagger may assign a wrong label (mainly Noun) to the verb denoting the event, which causes the extraction algorithm to discard it. Another possible source of error is the morphological analysis, because imperatives in Italian have the same form as verbs in the present indicative, therefore some misclassifications may arise: imperatives should be discarded from the event list, while verbs in the present indicative are generally included. Imperatives are quite frequent in our stories, because the characters tend to have short, direct interactions between them. In general terms, however, the implemented strategy seems to be effective in extracting factual events from children's stories.

5.3.5.2 Evaluation of anaphora resolution

The only Italian data set available with manually annotated anaphora is, to our knowledge, the one released for the coreference resolution task [39] at 2011 Evalita evaluation campaign. Therefore, we train our classifier on these data. We use the same training and test split for evaluation purposes, although we focus only on resolution of pronouns and zero anaphora, while the data set includes all coreferring entities. The maximum entropy classifier included in the OpenNLP suite was trained with 250 iterations.

The number of anaphoric elements in the test set is 515. The anaphora resolution module identifies 236 (46%) of them as anaphoric. The main source of error is the fact that our selection strategy is sometimes too greedy, discarding possible antecedents if they are more than 3 sentences distant from the anaphoric element. Also, agreement of person, noun and gender between the pronoun and the antecedent is a strong constraint, which is probably not robust enough to cope with errors in the morphological analysis.

Considering only the subset of these 236 anaphoric elements, the resolution algorithm achieves **P 0.43**, **R. 0.16** and **F1 0.23**. Most of the classification mistakes depend on the complexity of the documents in the text set. In fact, they often show convoluted syntactic structures, full of appositions and subordinating / coordinating constructions. Given the lower complexity of children's stories, we expect that anaphora resolution on this kind of texts will be less problematic. However, this requires the development of a domain-specific test set with stories being manually annotated with anaphoras. This is currently under development. Also the possibility to include in the training set other annotated data (e.g.[28]) is under consideration.

5.3.5.3 Evaluation of sentence simplification

Evaluating sentence simplification is not a trivial task, because a sentence may be simplified in several ways. Therefore, past approaches to the task have been usually evaluated by asking human raters to judge the correctness / plausibility of simplified sentences. In [7], for instance, the authors use Amazon's Mechanical Turk, asking three judges to indicate if the resulting simplified sentences are still correct in English.

In our case, the simplification process is constrained by the fact that *all and only* mandatory arguments of the verbs should be retained, which makes it possible to create a gold standard where all required participants in

the event are present and no much space is left to human interpretation. This allows us to use an evaluation approach which is stricter though more objective than post-hoc human judgment. We apply the Translation Error Rate (TER) measure [37], usually employed to evaluate automatic machine translation, which is computed as the number of edits needed to transform the output of a machine translation system into a reference translation, normalised by the number of words in the reference translation. The possible edit operations are the insertion, deletion and substitution of single words as well as changes in the position (shifts) of word sequences. We applied this measure to pairs made up of an automatically simplified statement (*hypothesis*) and a manually simplified statement containing the same event as the hypothesis (*reference*), using the TER-Plus evaluation tool.[7] Again, we do not perform a post-hoc evaluation, but we create the gold standard independently from the system output. For each event in the gold data created for the previous evaluation (Section 5.3.5.1) which was correctly extracted by our module, an annotator was asked to rewrite it in the present tense and remove all verb arguments which are not mandatory. For instance, in case of transitive verbs, only the subject and the object should be retained. In case of doubts, the annotator was allowed to look up the verb valence in the Sabatini – Colletti dictionary of Italian.[8] This manually simplified version of the sentences in the story are considered our *reference* in computing TER. Note that, while in machine translation evaluation several possible references are admissible, in our case only one reference is considered correct.

	Translation Error Rate
Baseline	0.73
Terence module: All Weights = 1	0.51
Terence module: Insertion = 0.5	0.41

Table 5.11: Evaluation of simplification process (the lower, the better)

The evaluation results are reported in Table 5.11. Note that TER ranges from 0 to 1, with 0 being a perfect simplification (i.e. the simplified version and the reference are identical, no edit operations are needed) and 1 being completely wrong. The baseline is obtained by removing all adjuncts (labeled as RMOD) to simplify the sentences. This means that we delete all arguments identified by the parser as optional (RMOD), without additional processing. We further compute TER with two different weighting schemes: in the first case, the weight of each edit operation is set to 1. In the second case, we set insertions to 0.5, meaning that if some argument is present in the system output and not in the reference, it is less penalized than the opposite case. In fact, since the reference contains all and only necessary information to describe an event, removing some words would probably produce an ungrammatical sentence. The opposite may lead to having a sentence only partially simplified but still grammatical.

In general, we notice that our simplification strategy combining lexical, syntactic and morphological information is more effective than a strategy purely based on syntactic information as produced by the parser. However, the current evaluation involves the simplification model in isolation. We leave the joint evaluation of the simplified sentences *with* resolved anaphora to future work.

[7] http://www.cs.umd.edu/~snover/tercom/
[8] http://dizionari.corriere.it/dizionario_italiano/

6 The Importance of Being Plausible: Preliminary Work for the New Generator

6.1 Introduction

During the TERENCE project life-cycle, we run an expert-based evaluation, documented in [12], to examine all the automatically generated textual games and fix errors right before the large-scale evaluation with the TERENCE learners. Having the learning material ready for this was the main goal of the expert-based evaluation. Still, the expert-based evaluation also gave useful feedback for the automated generation of textual games of WP4.

The first section below recaps the main results of the expert-based evaluation, so-far analysed, and that may affect the generation process. The second section documents the experimental work we set ourselves on in view of those results. Given the time and resource limits of the project, ours has to be considered preliminary work. However, we believe that the design of heuristics based on the expert-based evaluation results is promising for the future of improving the automated generation of games à la TERENCE.

6.2 Expert-based Evaluation of the Automated Generation

6.2.1 Introduction

The team working on the expert-based evaluation reported in [12] had experts of pedagogy and knowledgable of the TERENCE project. The most skilled expert evaluator acted as coordinator. We had nine experts for Italian games, all working with children, and two experts for English games. They were asked to evaluate the automatically generated textual games and fix them for the large-scale evaluation with learners. For instance, they were asked to perform the following three main operations:

operation 1: revision of

- the generated texts of game events or who questions, e.g., by fixing grammatical errors or exceeding length,
- the choices selected as central events, because either non-relevant or implausible,
- the choices selected as solutions, because either incorrect or implausible;

operation 2: deletion of games that could not be revised as above;

operation 3: creation of textual games whenever needed.

As reported in Chapter 5, some of the automatically generated game events and who questions were already evaluated by linguists, in particular, see the evaluation of simplification therein reported. In the case of the expert-based evaluation by pedagogy experts, the evaluation material was by far larger: it was made up of *all* the automatically generated smart games for all the TERENCE books of stories. Moreover, the main goal was different (that is, having the learning material ready for the large-scale evaluation with learners) and the timing for performing the evaluation was prohibitive—less than 2 months. That said, the procedure for the evaluation relied on human-judgment but tried to minimise biases that humans, albeit expert of pedagogy, may introduce in the evaluation.

To this end, specific guidelines for performing their work were presented and explained to all team members. An ad-hoc educator GUI, described in [3], was created to facilitate the work of pedagogy experts without specific ICT skills. Evaluators were trained to using the GUI and applying the guidelines for c.a one week. Once they were comfortable with the guidelines and GUI, each evaluator was randomly assigned the set of games of one story or more. Evaluators had to read their story first and edit the related games afterwards. In Italy, once an evaluator performed their work on the textual games for a story, the coordinator reviewed it. Finally, two evaluators were blindly assigned all revised textual games, and one evaluator was assigned all the created games. For a screenshot of the educator GUI that evaluators used for performing their work, see Figure 6.1. More precisely, this GUI has three main functionalities: one for (1) revising textual games, one for (2) creating textual games and another one for (3) deleting textual games.

The first functionality allowed evaluators to revise (1) the generated texts of game events or who questions, (2) the choices selected as central events, (3) revise the choices selected as solutions.

The second functionality allowed experts to create causality games, absent from the second release of D4.3 because of the lack of causal links in the annotations of WP3.

As for the third, evaluators used this functionality for deleting textual games that were irrelevant or very poorly generated.

For the evaluation in Italy, each expert was also instructed to fill in a structured diary in spreadsheet format, made up of 33 fields, specifying the changes made in *every* edited game and commenting on the reasons for the performed editing. This diary allowed for the monitoring of all activities and enables those in charge of the generation process to run a detailed analysis of the errors produced.

In the remainder, we sketch some of the Italian results of such expert-based evaluation that are relevant for the automated generation of textual games.

6.2.2 Main results and implications for the automated generation

In Italy, a total of 325 game instances were evaluated for 25 stories, out of which 250 character and time games were generated automatically (77%), and 75 were causality games created from scratch (30%).

In the following, we report the main results of the Italian expert-based evaluation and the suggested implications on the generation process design.

6.2.2.1 Development times

Results. The average revision times were estimated based on data reported by evaluators. The average time of each evaluator to revise a set of games for a story was equal to circa 76 minutes, i.e., approximately $12,6$ minutes per game instance, as recapped in Table 6.2. They were lower for revising who-games, and higher for revising time games. for a set of games, i.e., one game for each level, it was necessary to work for circa 76 minutes.

Solution	Pre-revision	Post-revision
AFTER	to thank	The inhabitants of the land of “pì” thank Jasmine
BEFORE	Louis leads the electrician the wires	*No change*
WRONG	All manage to split the fairly rubbish without difficult calculations needs	All manage to divide garbage in the right way

Table 6.1: Revision of a time game

Also according to experts’ feedback in diaries, time games were definitely the most challenging to work with for human beings. The highest proportion of revisions over all games (except causality games) was for before-after games (30%). According to all evaluators, the difficulty of such activity was that it required to locate the

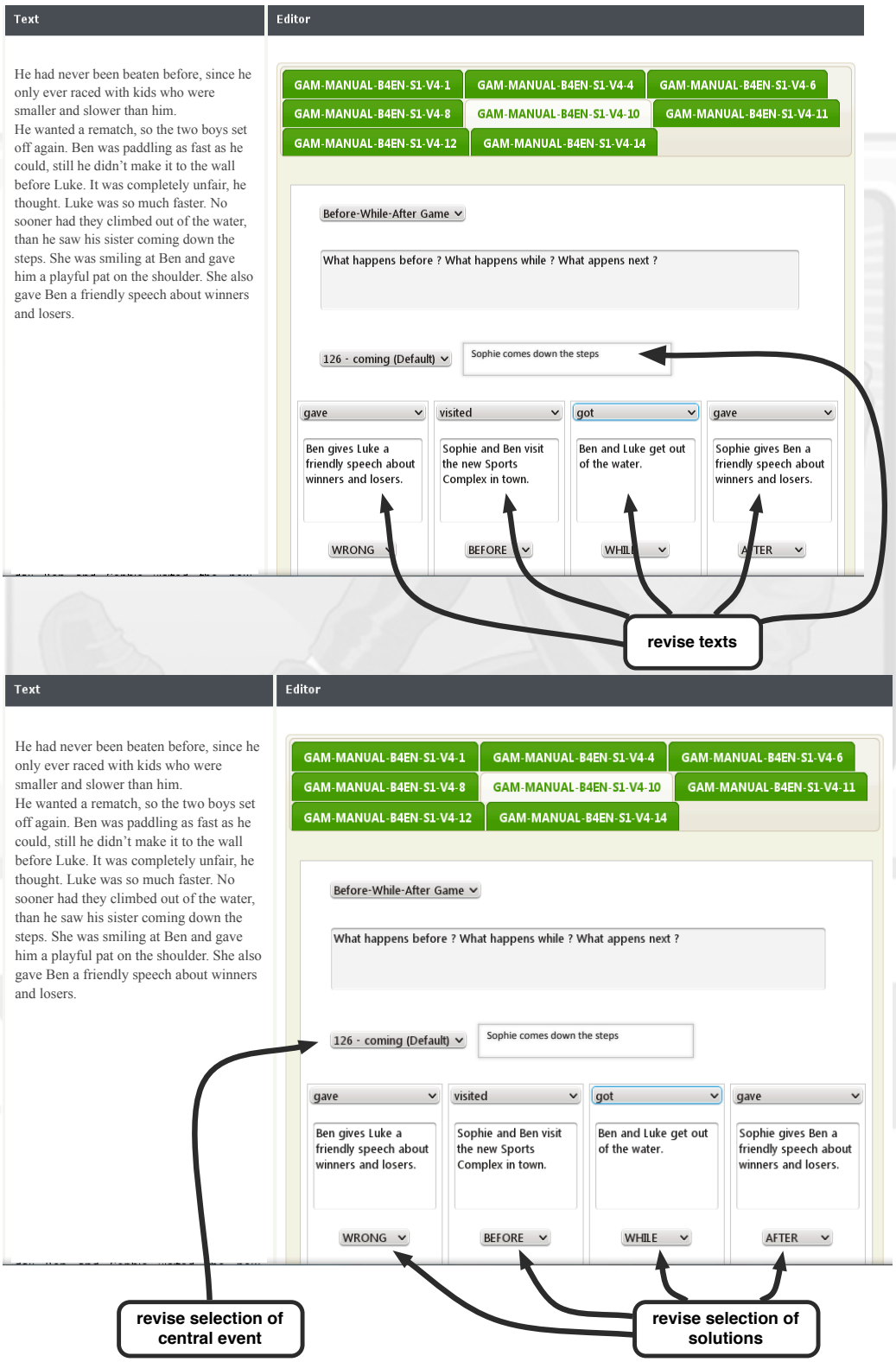


Figure 6.1: A screenshot of the educator GUI for editing textual games highlighting the allowed revisions

game events and their temporal relations across the entire story text. Table 6.1 shows example corrections performed on texts of a before-after time game.

GAME	n	%	Average time
WHO	25	10.00	10, 6
WHAT	34	13.60	12
BEFORE-AFTER	74	29.60	12, 8
BEFORE-WHILE	41	16.40	12, 8
WHILE-AFTER	42	16.80	12, 8
BEFORE-WHILE-AFTER	34	13.60	14, 8
Total	250	100.00	12, 7

Table 6.2: Details about the revision of textual games

The work of creating from scratch causality game instances was instead longer, according to data reported by evaluators. Overall, 75 causality game instances were created. The average time spent for their development was equal to 23 minutes per game instance. about 30 minutes for each causality game instance.

Possible implications. Such results seem to suggest that generating automatically textual games may help in reducing development costs.

6.2.2.2 Ranking strategy

Results. Only in 15 out of 250 cases (6%), it was necessary to select a different central event than the automatically generated one. Solutions of games are in number of 3 except for before-while-after games that have 4 solutions. In total, the evaluators performed 140 changes of the automatically selected solutions, which yields c. slightly less than 18% revisions of selections of solutions, wrong or correct.

Possible implications. Such results seem to indicate that the ranking strategy for selecting the central events and solutions, explained in Chapter 4, pays off.

6.2.2.3 Plausible wrong solutions

Results. Out of 140 changes of selection of solutions, those for wrong choices, not sufficiently plausible as distractors, were in number of 54. Considering that the majority of games (c. 80%) have at least 2 correct solutions and only 1 wrong solution, that means that the majority of the performed changes of selections were for wrong solutions. Such result is also confirmed by observations of experts traced in diaries as well as during the large-scale evaluation. Moreover, the same remark was put forward in the first expert based evaluation of the TERENCE project, recapped in [14]. See also the evaluation results of linguists for simplification in Chapter 5.

Possible implications. Such results call for a remedy action for generating wrong solutions that work as more plausible distractors.

6.2.2.4 Co-reference

Results. In 180 cases out of 250 (72%), the revisions were for the text generated by the NLG module for the central event; similar figures are for the text generated for solutions. In particular, the revisions of who-games sometimes required choosing different solutions or rewriting them due to errors introduced in the annotation process of WP3, mainly for resolving co-references. For instance, in a WHO game, the generated question

was “Who are the real heroes?”. The generated correct solution was “the blue phins” in place of “the dolphins”. necessary corrections, we had to (i) choose a new character for each solution, and (ii) verify that it was the correct/wrong solution.

Possible implications. Such results indicate that co-reference resolution needs improved technologies for Italian. This is in-line with the evaluation results of linguists for anaphora resolution, reported in Chapter 5.

6.3 The Novel Generation of Less Implausible Distractors

As highlighted above, one of the main changes performed by experts were for the wrong choices, not sufficiently plausible as distractors. Therefore, in the remaining life time of the project, albeit short, we set ourselves on a new tack: trying to generate less implausible distractors for all games, except who games, according to the revisions of experts and their comments reported in diaries. The remaining of this section documents the first heuristics that were designed on top of such results, and the novel NLG module for generating game events according to such heuristics, where game events are simplified sentences that label pictures of events in all games except WHO-games, see also Chapters 4 and 5. The work resulted from a combined effort of FBK-irst, LUB an UnivAQ.

6.3.1 The Expert-based Heuristics for Game Events

Each type of heuristics for the novel generation of game events consider an event E in the story and generate (1) a simplified sentence for the correct solutions and (2) another sentence for the wrong solutions of games. The so-far developed heuristics are described below.

6.3.1.1 $H1$ heuristics

Let X be an ENTITY X that is a participant (via ENTITY_MENTION and REFERS TO) in E.

Choose an ENTITY Y in the story that is different than ENTITY X, and check that it is not a participant in an event E’ that is identical to E, with the following S1 and S2 heuristics (in decreasing priority).

S1. ENTITY Y has

- the same ENTITY TYPE as ENTITY X
- STORY ROLE equal to MAIN or SECONDARY

S2. If none according to S1 then ENTITY Y has the same ENTITY TYPE as ENTITY X

If there is such an ENTITY Y then generate the WRONG sentence for EVENT E with ENTITY Y instead of ENTITY X.

Example 3 *CORRECT SOLUTION sentence for EVENT E: “Ernesta salta sulla bicicletta”. WRONG SOLUTION sentence for EVENT E: “Gli astronauti saltano sulla bicicletta”.*

6.3.1.2 $H2$ heuristics

Negate EVENT E in the generated sentence, if not already negated. Else, make it assertive.

Then generate the WRONG sentence for negated/asserted EVENT E.

Example 4 *CORRECT SOLUTION sentence for EVENT E: “Ernesta salta sulla bicicletta”. WRONG SOLUTION sentence for EVENT E: “Ernesta non salta sulla bicicletta”.*

6.3.1.3 $H3$ heuristics

Add a temporal anchor (e.g., time expression or subordinate) to EVENT E in the generated sentence that is different than any temporal anchor in the story/in the event E.

Then generate the WRONG sentence for EVENT E with 'wrong' temporal anchor.

Example 5 *CORRECT SOLUTION sentence for EVENT E: "Un pomeriggio d'estate [. . .]. Ernesta salta sulla bicicletta". WRONG SOLUTION sentence for EVENT E: "Ernesta salta sulla bicicletta una mattina d'inverno".*

6.3.2 NLG technologies

We extended the framework for game sentence and WHO-question generation reported in Chapter 5 with a new module to generate, for each factual event,

- a game sentence, that is, a simplified sentence that describes the event,
- and three possibly wrong alternatives, according to the aforementioned heuristics.

Compared to the previous version of the system, the new one takes in input an XML file in TAF. This can be either a "gold" file, with manual annotation of events, participants and temporal links or causal links, or an XML automatically generated and enriched as described in Chapter 3. In the latter case, we expect the outcome of the simplification and generation task to be less precise due to error introduced in the automated annotation process. Alternatively, a user can also give in input an event ID, and the module will generate only the simplified sentence and the three heuristics for this specific event.

The output of the module will be an XML file that, for each event, lists the original sentence, the simplified one, and the three possibly wrong alternatives. An example output is reported below:

```
<EVENT id="2">
<original_sentence id="2">
Sul razzo, viaggiavano lei e la cagnolina Chiazza.
</original_sentence>

<simplified_sentence>
Sul razzo, viaggiano Ernesta e la cagnolina Chiazza.
</simplified_sentence>

<H1 type="subj">
Sul razzo, viaggia Nerino.
</H1>

<H2>
Pochi giorni prima sul razzo, viaggiano Ernesta e la cagnolina
Chiazza.
</H2>

<H3>
Sul razzo, non viaggiano Ernesta e la cagnolina Chiazza.
</H3>
</EVENT>
```

In the simplified sentence, the event tense has been changed into present. In the first heuristic (<H1>), a wrong subject has been inserted and the verb has been conjugated accordingly. In the second one (<H2>), a wrong temporal expression has been inserted. In the third one (<H3>), the event has been negated.

More specifically, given a TAF file, the new NL module performs the following steps in sequence.

- The TAF file is processed in order to extract the list of temporal expressions, entities and events annotated in the document. It is also processed with the TextPro NLP suite to obtain morphological information on each token in the story.
- For each event e_i, the list of participants is extracted.
- Since each participant p_i is connected through an ID to an entity (see step 1), the *story_role* and *entity_type* attributes are retrieved from the entity and assigned to p_i.
- For each participant p_i, the module also retrieves the gender and number. Here several strategies are applied according to the entity_type assigned to p_i (e.g. if it is a proper name, a common noun, etc.).
- We simplify the sentence containing the event e_i. This is performed as described in D4.2, without taking into account the additional information annotated in the XML file
- We apply the $H1$ heuristics to the simplified sentence: replace its subject with another one taken from the entity list extracted from the whole story. After the replacement, the verb number must be checked and changed if necessary. If the verb is transitive, a variant of this heuristic can be applied to the object.
- We apply the $H2$ heuristics: if the simplified sentence does not contain a temporal expression, add a timex taken from the list created in Step 1. If there is already one, replace it with another (the first of the list).
- We apply the $H3$ heuristics: the event in the simplified sentence is negated. While negation in Italian usually precedes a verb, its position changes if a pronoun appears in the sentence. All this must be taken into account when applying $H3$.

Some critical issues concern verb constructions in which the subject gender and number are marked on the verb, for instance passive forms and past forms of verbs of movement. In these cases, replacing the existing subject with a new one implies *i)* checking the gender and number of the new subject, *ii)* checking gender and number of the verb, *iii)* checking if they are compliant and, if not, *iv)* modifying the verb construction.

7 Conclusions

The current document serves to describe the WP4 generation process of textual games, that is, textual components of smart games in XML, starting from the stories annotated in WP3. The generated textual games, in turn, become input for the visualisation module of WP6 that assembles the graphical and textual components of the TERENCE learning material, automatically.

One of the key features of TERENCE is its design life-cycle, based on the user centred design, that is iterative and incremental. Evaluations are also used to inform the design or revise it. The user centred design has also evaluation methods that involve domain experts. The last expert-based evaluation reported in [12] was used to assess the second-release of the generation of textual games.

In view of the results of the expert-based evaluation with pedagogy experts, we can claim that the generation process had, in its second release, the following main limitations:

1. CLINKs are problematic for current annotation technologies, making it impossible to generate causality games;
2. co-reference resolution is problematic, in particular for Italian, affecting the quality of the generated WHO-games,
3. wrong solutions selected by the reasoning module were not often plausible.

Improvements to the final release of the annotation module of WP3 tried to limit the first problem. For the second, the existing limitations could not be tackled in the remaining life-time of the project. Routes to possible solutions to the third problem were inspected. Performing such work meant analysing the rich set of structured data collected by Italian evaluators in diaries for the second expert-based evaluation, and then starting the design of first heuristics for improving the plausibility of wrong choices. The main results so-far analysed for improving the generation process and the resulting preliminary heuristics are briefly sketched in Chapter 6. At the time of writing, this work is still on-going.

Bibliography

[1] E Adams. *Fundamentals of Game Design*. New Riders, 2010.

[2] J. F. Allen. Maintaining Knowledge about Temporal Intervals. *ACM Comm.*, 26:832–843, 1983.

[3] M. Alrifai. Global System Report, Deliverable 2.4. Technical report, TERENCE project, 2012.

[4] M. Alrifai and R. Gennari. Game Design, D2.3. Technical report, TERENCE project, 2012.

[5] Krzysztof R. Apt. *Principles of Constraint Programming*. Cambirdge University Press, 2003.

[6] Gianni Barlacchi and Sara Tonelli. ERNESTA: A sentence simplification tool for children's stories in Italian. In *Proceedings of the 14th International Conference on Intelligent Text Processing and Computational Linguistics (CiCLing 2013) (to appear)*, Samos, Greece, 2013.

[7] Jan DE Belder and Marie-Francine Moens. Text simplification for children. In *Proceedings of the SIGIR 2010 Workshop on Accessible Search Systems*, 2010.

[8] Delphine Bernhard, Louis de Viron, Veronique Moriceau, and Xavier Tannier. Question Generation for French: Collating Parsers and Paraphrasing Questions. *Dialogue and Discourse*, 3(2):43–74, 2012.

[9] Steven Bethard. State of the Art and Design of Novel Annotation Languages and Technologies. Technical Report D3.1, TERENCE project, 2011.

[10] Tommaso Caselli, Valentina Bartalesi Lenzi, Rachele Sprugnoli, Emanuele Pianta, and Irina Prodanof. Annotating Events, Temporal Expressions and Relations in Italian: the It-TimeML Experience for the Ita-TimeBank. In *Proceedings of LAW V*, Portland, Oregon, USA, 2011.

[11] V. Cofini, F. de la Prieta, T. Di Mascio, R. Gennari, and P. Vittorini. Design Smart Games with Context, Generate them with a Click, and Revise them with a GUI. *Advances in Distributed Computing and Artificial Intelligence Journal*, 3, Dec. 2012 2012.

[12] T. Di Mascio. Expert-based Evaluation. 2nd Release. public, TERENCE project, 2013.

[13] T. Di Mascio, R. Gennari, and P. Vittorini. D7.3. Small-scale Evaluation Results. Technical report, TERENCE project, 2012.

[14] T. Di Mascio, R. Gennari, and P. Vittorini. Expert-based Evaluation. 1st Release. public, TERENCE project, 2012.

[15] T. Di Mascio, R. Gennari, and P. Vittorini. D7.4. Large-scale Evaluation Results. Technical report, TERENCE project, 2013.

[16] Z. Gantner, M. Westphal, and S. Wölfl. GQR - A Fast Reasoner for Binary Qualitative Constraint Calculi. In *AAAI'08 Workshop on Spatial and Temporal Reasoning*, 2008.

[17] R. Gennari. State of the art and design of novel intelligent feedback, Deliverable 4.1. Technical report, TERENCE project, 2011.

[18] R. Gennari. Generation Service and Repository of Textual Smart Games. Technical report, TERENCE project, 2012.

[19] Rosella Gennari. Temporal Constraint Programming: a Survey. *CWI Quarterly Report*, 1998.

[20] Michael Heilman and Noah A. Smith. Extracting Simplified Statements for Factual Question Generation. In *Proceedings of QG2010: The Third Workshop on Question Generation*, Pittsburgh, PA, USA, 2010.

[21] Morris S.Y. Jong, Jimmy H.M. Lee, and Junjie Shang. *Reshaping Learning*, chapter Educational Use of Computer Games: Where We Are, And What's Next. Springer, 2013.

[22] A. Krokhin, P. Jeavons, and P. Jonsson. Reasoning about Temporal Relations: The Tractable Subalgebras of Allen's Interval Algebra. *Journal of ACM*, 50(5):591–640, 2005.

[23] A. Lavelli, J. Hall, J. Nilsson, and J. Nivre. MaltParser at the EVALITA 2009 Dependency Parsing Task. In *Proceedings of EVALITA Evaluation Campaign*, 2009.

[24] A.K. Mackworth. Consistency in Network of Relations. *Artificial Intelligence*, 8, 1977.

[25] N. Nebel and H.-J. Bürckert. Reasoning About Temporal Relations: A Maximal Tractable Subclass of Allen's Interval Algebra. *Journal of ACM*, 42(1):43–66, 1995.

[26] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ACL '02, pages 311–318, Stroudsburg, PA, USA, 2002. Association for Computational Linguistics.

[27] Emanuele Pianta, Christian Girardi, and Roberto Zanoli. The TextPro tool suite. In *Proc. of the 6th Language Resources and Evaluation Conference (LREC)*, Marrakech, Morocco, 2008.

[28] Massimo Poesio, Rodolfo Delmonte, Antonella Bristot, Luminita Chiran, and Sara Tonelli. The VENEX corpus of anaphora and deixis in spoken and written Italian. Technical Report Essex NLE Technical Note 2003-1, University of Essex, 2006.

[29] J. Pustejovsky, J. Castano, R. Ingria, R. Saurí, R. Gaizauskas, A. Setzer, G. Katz, and D. Radev. TimeML: Robust specification of event and temporal expressions in text. In *IWCS-5 Fifth International Workshop on Computational Semantics*, 2003.

[30] J. Pustejovsky, P. Hanks, R. Sauri, A. See, R. Gaizauskas, A. Setzer, D. Radev, B. Sundheim, D. Day, L. Ferro, et al. The TimeBank corpus. In *Corpus Linguistics*, volume 2003, page 40, 2003.

[31] J. Pustejovsky, K. Lee, H. Bunt, and L. Romary. ISO-TimeML: An International Standard for Semantic Annotation. In *Proceedings of the Fifth International Workshop on Interoperable Semantic Annotation (ISA-5)*, 2010.

[32] J. Pustejovsky, J. Littman, and R. Saurí. Arguments in TimeML: events and entities. *Annotating, Extracting and Reasoning about Time and Events*, pages 107–126, 2007.

[33] J.M. Santos, L. Anido, M. Llamas, L.M. Álvarez, and F.A. Mikic. *Computational Science*, chapter Applying Computational Science Techniques to Support Adaptive Learning, pages 1079–1087. Lecture Notes in Computer Science 2658. 2003.

[34] Advaith Siddharthan. Complex lexico-syntactic reformulation of sentences using typed dependency representations. In *Proceedings of the 6th International Natural Language Generation Conference (INLG 2010)*, Dublin, Ireland, 2010.

[35] Advaith Siddharthan. Text Simplification using Typed Dependencies: A Comparision of the Robustness of Different Generation Strategies. In *Proceedings of the 13th European Workshop on Natural Language Generation*, 2011.

[36] Karin Slegers and Rosella Gennari. State of the Art of Methods for the User Analysis and Description of Context of Use. Technical Report D1.1, TERENCE project, 2011.

[37] Matthew Snover, Bonnie Dorr, Richard Schwartz, Linnea Micciulla, and John Makhoul. A study of translation edit rate with targeted human annotation. In *Proceedings of Association for Machine Translation in the Americas*, 2006.

[38] Toby J. Teorey, Dongqing Yang, and James P. Fry. A Logical Design Methodology for Relational Databases using the Extended Entity-relationship Model. In *ACM Computing Surveys*. ACM, 1986.

[39] Olga Uryupina and Massimo Poesio. Evalita 2011: Anaphora Resolution Task. In *Proceedings of EVALITA Evaluation Campaign*, 2011.

[40] van Beek, P. Reasoning about Qualitative Temporal Information. *Artificial Intelligence*, 58(1-3):297—326, 1992.

[41] Marc Verhagen, Roser Sauri, Tommaso Caselli, and James Pustejovsky. SemEval-2010 Task 13: TempEval-2. In *Proceedings of the 5th International Workshop on Semantic Evaluation*, pages 57–62, Uppsala, Sweden, July 2010. Association for Computational Linguistics.

[42] P. Vittorini. D6.2. Visualization Module. Technical report, TERENCE project, 2013.