The TERENCE Smart Games: Automatic Generation and Supporting Architecture

Fernando De la Prieta[1], Tania Di Mascio[2], Rosella Gennari[3], Ivana Marenzi[4], and Pierpaolo Vittorini[5]

[1] U. of Salamanca, Deparment of Computer Science, Plaza de la Merced s/n, 37008, Salamanca, Spain
[2] U. of L'Aquila, DIEI, Via G. Gronchi 18, 67100, L'Aquila, Italy
[3] Free U. of Bozen-Bolzano, P.zza Domenicani 3, 39100 Bolzano, Italy
[4] L3S, University of Hanover, Germany
[5] U. of L'Aquila, Dep. of MeSVA, V.le S. Salvatore, Edificio Delta 6, 67100, L'Aquila, Italy

Abstract. TERENCE is an FP7 ICT European project that is developing an adaptive learning system for supporting poor comprehenders and their educators. Its learning material are stories and games. The games are specialised into smart games, which stimulate inference-making for story comprehension, and relaxing games, which stimulate visual perception and not story comprehension. The paper focuses on smart games. It first shows the current prototypes, then it describes the TERENCE system architecture, thus it delves into the generation of smart games instances, by highlighting the role of the constraint-based module therein. Finally, it ends with short conclusions about the planned improvements.

Keywords: Game frameworks, serious games architectures, reusability, user centred design

1 Introduction

Nowadays, circa 10% of young children are estimated to be poor (text) comprehenders [14]: they are proficient in word decoding and other low-level cognitive skills, but they show problems in deep text comprehension [8, 15]. TERENCE [18] is a European ICT multidisciplinary project, for the technology enhanced learning (TEL) area, that is developing the first *adaptive learning systems* (ALS, [6, 7]) for improving the reading comprehension of 8–10 year old poor comprehenders.

The learning material of TERENCE is made of stories and games, written in the two languages of the project, Italian and English. The learning material was designed following the user centred and evidence-based design, see [9]. The models for the learning material and learners of the system are in [2], and the first adaptation rules are in [3], whereas [4] explains how the models and learner model, in particular, stem from an extensive context of use and requirement analysis [17, 11]. In particular, the TERENCE games are distinguished by the evidence-based pedagogical plan into smart and relaxing. Smart games in TERENCE consist of reasoning tasks about the characters and events of a story, as well as on their relations, and are designed on effective pedagogical interventions by educators as well as pedagogy and therapy experts for assessing

and stimulating story comprehension, see [10]. As such, smart games are cognitively demanding for the learner, whereas relaxing games allow the learner to have breaks.

This paper, which is of a technical nature, focuses on the automatic generation of smart games using the TERENCE framework, as well as on the underlying architecture. Sec. 2 introduces background concepts, so as to make the reader grasp what the smart games are like. With such background and intuitions, the paper shows prototypes of smart games (Sec. 3), then delves into the TERENCE architecture and the methodology for the automatic generation of smart games from stories (Sec. 4).

2 Background: Pedagogically-driven Reading Interventions for Smart Games

The TERENCE smart games mainly serve for stimulating the learner in enhancing reading comprehension. How the design of the games is based on effective reading interventions, selected by the TERENCE pedagogical plan, is described in [12, 10]. In brief, the plan guides the process of deep text comprehension by proposing increasingly demanding tasks: firstly, it makes the learner reason about the characters that are in the story, then about the events that take place in the story, hence about temporal relations among the events, and finally about causal-temporal relations among events. For instance, let us consider the following story excerpt.

> *Perla took a book from the library shelf [. . .] She watched the clowns in the circus. They were throwing cakes here and there [. . .] The little girl ducked just in time to avoid the flying cake.*

Typical reading interventions, selected by the plan, ask the reader to infer that:

(a) "Perla watches the clowns in the circus" (the anaphoric resolution "She" $\rightarrow$ "Perla" is needed),
(b) "Perla takes a book" and "Perla watches the clowns in the circus" (though, for instance, Perla does not throw cakes),
(c) "Perla takes a book from the library shelf" before "Perla watches the clowns in the circus", and
(d) "Perla ducks" because "The clowns throw cakes".

The TERENCE smart games embed such interventions in game-like environments. They are organised as in the taxonomy in Fig. 1. The children of the root are so defined:

(1) *characters*: smart games concerning characters, namely, who an agent of a story's event is (who-game, e.g. the example in point (a) above), what a character does in the story (what-game, e.g. the example in point (b) above);
(2) *time*: smart games for reasoning about temporal relations between events of the story, purely sequential (before/after-game, e.g. the example in point (c) above) or not (all the others);
(3) *causality*: smart games concerning causal-temporal relations between events of the story, namely, the cause of a given event (cause), the effect (effect), or the cause-effect relations between two events (cause/effect-game, e.g. the example in point (d) above).

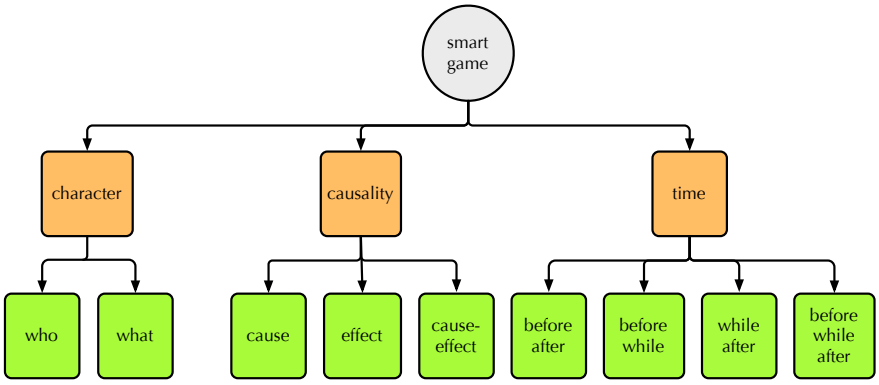

Fig. 1. Hierarchy of the TERENCE smart games

3 Prototypes of Smart Games

For giving the reader a concrete feeling of how reading interventions were turned into smart games in TERENCE, this brief section shows prototypes of smart games. However, we only pause on the features of the prototypes that are relevant for explaining the generation of smart games, which is the focus of the remainder of this paper.

Fig. 2 and 3 show two prototypes, at different states. In all prototypes, the interface is split in two areas. The top area contains the points for the game, and the avatar of the learner. The bottom area is divided into two parts. One part contains the main question. The other part shows the choices available to the learner in the current state. The feedback can be different, see [12]. For instance, in the shown prototypes, the reader can see a consistency feedback, in the form of a yes/no visual message, displayed on top of the available choices; an explanatory feedback for wrong solutions is shown in the form of a red light bulb between two events in case the learner places them in a temporal relation that is inconsistent with the story.

Fig. 2. Screenshots of a what game prototype.

Now, more in details, the left screenshot in Fig. 2 shows an intermediate state of a multiple-choice *what* game: the question posed to the learner is "What happens in the story?". The learner has to choose and drag one of the choices as a key, and see if it opens (or not) the cupboard. The right screenshot in Fig. 2 shows the feedback concerning the (in)consistency of the choice with the story in the form of a *no* visual message. If the resolution is correct, the locker opens and a reward drops. Notice that the visual metaphor are adapted to the age of the learner, e.g., a locker is used for 9-11 learners whereas a cupboard is used for 7–9 learners.

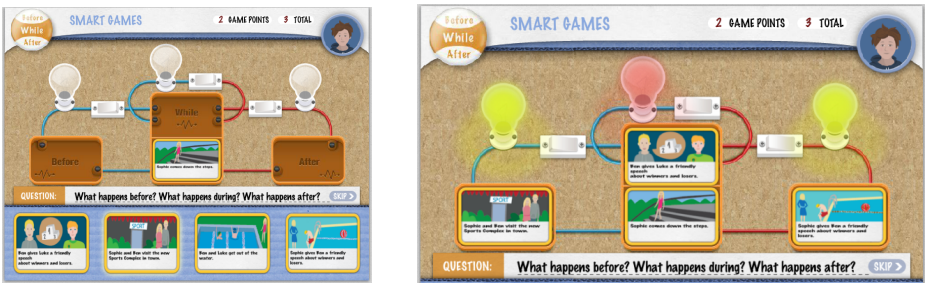

Fig. 3. Screenshots of a time game prototype.

The left screenshot in Fig. 3 shows the initial state of an ordering time game: the learner has to establish before, while and after relations with the event displayed in the centre of the top area. To do so, the learner has to choose and drag events from the bottom area, and drop them into the appropriate empty container in the top area. The right screenshot in Fig. 3 shows an explanatory feedback in the top area: correctly placed relations are shown with yellow greens; the wrongly placed relation is signalled with a red bulb. Here as well, a different metaphor is used with younger learners, with water and mill-wheels in place of electricity and light bulbs.

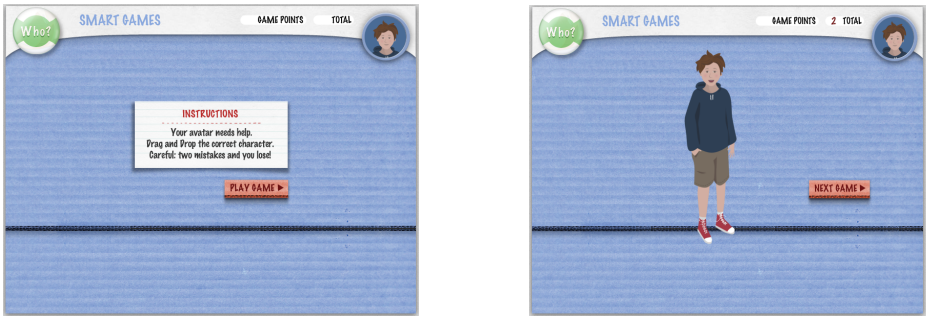

Fig. 4. The smart games activity: instructions and avatar feedback.

Finally, the left screenshot in Fig. 4 shows the instructions presented to a learner before a smart game starts. In order to maximise the chances that the learner reads the instructions, five seconds of a fake loading activity are added. Afterwards, a "Play game" button is shown. The right screenshot in Fig. 4 shows that, after each game, the avatar gives a further feedback to the learner: in case the learner solves the game correctly, an happy avatar is shown; otherwise a sad avatar is displayed.

4 System Architecture

Fig. 5 shows the main components of the TERENCE system. Although the figures display them in layers, all work independently and the information is exchanged through RESTful web services. In this manner, high cohesion and low linkage is ensured, as well as the re-usability of each component.

An overview of the whole system, organized in layers, is shown below.

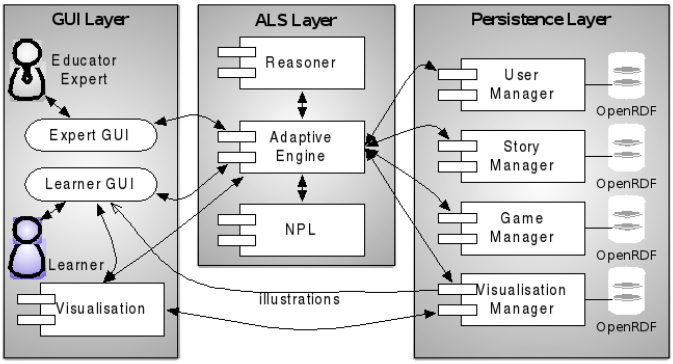

Fig. 5. The architecture.

GUI Layer. This functional layer is divided into three main components:

- the Educator GUI, that is a Rich Internet Application (RIA) built on top of the Vaadin framework [19] that provides the necessarily facilities to manage the TERENCE learning material;
- the Learner GUI, that is designed to be used in normal PCs as well as tablets, and provides a Flash environment where learners read stories and play with the TERENCE smart games;
- the Visualisation module that is in charge of preparing the visual information (e.g., illustrations, templates) that are used in the Learner GUI. For instance, concerning the smart game in Fig. 2, the visualisation module provides the background image, the boxes, the question and the avatar.

ALS Layer. This layer contains the core system components:

- the NPL module, which is in charge of annotating the stories with tags useful for generating the TERENCE smart games;
- the Reasoner module that (i) checks the consistency of the annotations of each story, (ii) enriches the story with further temporal annotations missed during the annotation process, and (iii) automatically generates smart game instances from the enriched annotations;
- the Adaptive Engine, which orchestrates the other components and establishes adaptive sessions for each learner, by taking into account their profiles.

Persistence Layer. This layer is divided into four components, that is, the User, Story, Game and Visualisation Manager, each with its own repository. The repositories are based on RDF ontologies. This schemaless model has two main advantages: it enables the possibility of evolving the system (adding more functionalities or refining existing ones) without large efforts, and allows for further reasoning services about the stored data.

Among the various functionalities of the system, we focus on the management of the smart games, and specifically on how the smart games are generated. Fig. 6 shows an overview of the entire process.

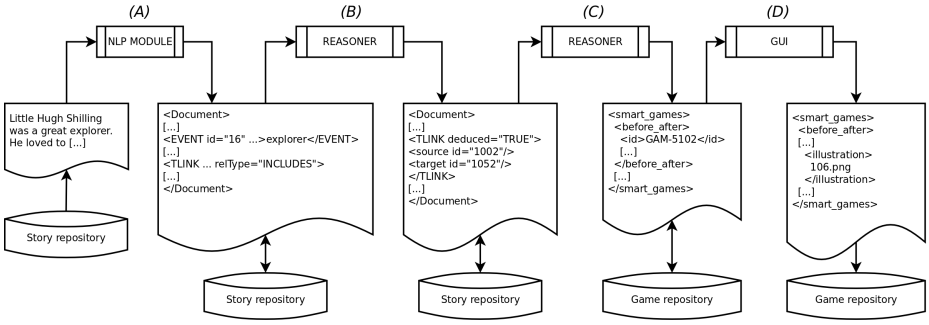

Fig. 6. Smart games generation process

Phase A. Firstly, from a story text contained in the story repository, the NPL module generates a story annotated with a variant of the TimeML language that was extended in [16] with tags for information that is relevant for the TERENCE smart games, e.g., the ENTITY and CLINK tags, that aim, respectively, to represent the entity related to an event, and the causal-temporal relations between two events. The annotated story is then stored in the same repository.

Phase B. Then, the reasoner checks the consistency of the annotations, detects the eventual temporal inconsistencies, and enriches the annotations by adding deduced temporal relations as further TLINK tags. This new consistent and enriched story is also stored in the story repository.

Phase C. Afterwards, starting from the enriched story, the reasoner module generates automatically instances of smart games that are stored in the game repository.

Phase D. Finally, a manual revision of the generated smart game instances takes place, where the related visuals (e.g. background illustrations, buttons) are also specified.

In the following, we delve into phase C of the process. Starting from a story s, annotated and then enriched as explained above, the smart games generation goes as follows:

Algorithm 1 The algorithm for generating smart game instances.

Require: story s, number of events n, number of games k
1: **foreach** event e **in** s **do**
2: generate all types of games for event e
3: **end for**
4: **sort** events
5: **keep** the games for the first *n* events
6: **reduce** the total number of games to fixed number *k*

Algorithm 1 initially iterates among all events, tagged with the EVENT tag in story s. Iteratively, an event e is selected as the *fixed event* for the generation process. Then, the algorithm generates instances of smart games with e and other events (lines 1–3).

For example, let us consider a time before-after games, shown in Fig. 3. The fixed event is displayed as the central even in the figure. Then the algorithm, using specific heuristics, finds an event that happens before the fixed event, and one that happens after the fixed event, and a further event that does not happen before or after the fixed event in the story. Accordingly, all possible before-after games for the fixed event e are generated as follows:

Algorithm 2 The before-after games generation algorithm.

Require: event e, story s, story $s_o \neq s$
1: **foreach** tlink t_1 **in** s, that has e as target **do**
2: **foreach** tlink t_2 **in** s, that has e as source **do**
3: **select** a random event w from story s_o
4: **create** a before/after game that has:
5: e as fixed event
6: the source event of t_1 as before
7: the target event of t_2 as after
8: w as wrong event
9: **end for**
10: **end for**

Algorithm 2 initially selects all pairs of temporal relations, that is, TLINKs that have e as target/source in the enriched annotated story. For each pair $[t_1, t_2]$, it first tries

to select an event that is related with a while relation with e. In case of failure, it then selects a fake event w that does not happen in the story. For further details on how each type of game is generated, please refer to [1].

After all possible games are generated, Algorithm 1 produces an ordered list of fixed events (line 4) according to the following heuristics. Given two fixed events e_1 and e_2, in order to decide if $e_1 > e_2$, we compare the related number of generated games, weighting these according to their difficulty level, established by the stimulation plan. In other words, $e_1 > e_2$ if the number of cause-effect games for e_1 is higher than for e_2. If equal, $e_1 > e_2$ if the number of effect games for e_1 is higher than for $e2$. If still equal, we compare the number of cause games, etc.

After the ordering, two pruning phases take place. The first keeps only the games for the first n fixed events (line 5) in the ordered list.The second pruning is concerned with the total number of smart games, reduced to a fixed number k. For each game of a given level, the algorithm keeps with different reasoning complexity, e.g., before-after games with both "deduced" events, implicit in the text, as well as who-games and what-games with both "protagonists" and "secondary" characters as agents (line 6). For further details on how games are removed from the list, please refer to [13].

Fig. 7 shows an excerpt of an XML document that contains the textual instance of a before-after game (lines 4–24), conforming to the conceptual models defined in [5], generated as described above. The game contains the fixed event (line 22), and three choices representing the event that happens before the fixed event (lines 17–21), the event that happens after the fixed event (lines 12–16), and the wrong choice (lines 7–11). Note that each choice contains:

- its correctness, i.e. if the choice is (i) always correct, (ii) correct if selected as before, after, or while, (iii) correct if selected as cause or effect; for instance, line 13 states that the second choice is correct if selected as the event that happened after the fixed event;
- the event it represents (e.g. line 14 for the second choice), and
- the text that will be shown to the learner (e.g. line 15 for the second choice).

Note that:

- the textual sentences/questions contained in the example (i.e., instructions, textual labels of the illustrated events) require manual rewritings so to become comprehensible to learners. For instance, the textual label for the game fixed event (line 23) is simply "asked" [1]. Since not enough explicative, it must be manually rewritten, e.g., into "Little Hugh asked the little boy why the streets were empty". However, it is planned that the next release of the NLP module will implement an automated functionality that generates such textual sentences;
- the heuristic for selecting the relevant events has still to be assessed and improved, e.g., by adjusting the heuristic through a comparison of the results with manually selected events.

[1] So far, the field is always initialised with the event text.

```
<?xml version="1.0" encoding="UTF-8"?>
<smart_games>
 ...
  <before_after>
    <id>GAM-5102</id>
    <type>TERENCE_SG_BEFORE_AFTER</type>
    <choices>
      <correct>WRONG</correct>
      <event id="550" modality="NONE" morphology="NONE"> stand </event>
      <text> stand </text>
    </choices>
    <choices>
      <correct>AFTER</correct>
      <event id="644" modality="NONE" morphology="NONE"> blackened </event>
      <text> blackened </text>
    </choices>
    <choices>
      <correct>BEFORE</correct>
      <event id="166" modality="NONE" morphology="NONE"> looked </event>
      <text> looked </text>
    </choices>
    <event id="558" modality="NONE" morphology="NONE"> asked </event>
    <text> asked </text>
  </before_after>
 ...
</smart_games>
```

Fig. 7. An example of a textual instance of a smart game

5 Conclusions

The paper described the architecture supporting the management of the TERENCE smart games, starting from their generation until their usage in the prototypes currently developed. The design of the current prototypes is being revised in light of the results of small-scale usability evaluations, expert based evaluations by psychologists and interaction designers. On-going work is considering the adaptation of the feedback to the learner, the automatic generation of the explanatory feedback by the reasoning module, improvements in the automatic generation of the textual instances of the smart games, and improvements in the performances of the overall architecture.

Acknowledgments. This work was supported by the TERENCE project, funded by the EC through FP7 for RTD, ICT-2009.4.2. The contents of the paper reflects only the authors' view and the EC is not liable for it. The authors thank MOME for producing the images of events in snapshots of games. Gennari work was also funded through the CRESCO and DARE projects, financed by LUB and Province of Bozen-Bolzano.

References

1. Alrifai, M., Gennari, R.: Deliverable 2.3: Game Design. Tech. Rep. D2.3, TERENCE project (2012)

2. Alrifai, M., Gennari, R., Tifrea, O., Vittorini, P.: The Domain and User Models of the TERENCE Adaptive Learning System. In: Vittorini, P., Gennari, R., Marenzi, I., de la Prieta, F., Corchado, J.M. (eds.) Proc. of the ebTEL 2012 workshop co-located with PAAMS 2012. Soft Computing, Springer (2012)
3. Alrifai, M., Gennari, R., Vittorini, P.: Adapting with Evidence: the Adaptive Model and the Stimulation Plan of TERENCE. In: Vittorini, P., Gennari, R., Marenzi, I., de la Prieta, F., Corchado, J.M. (eds.) Proc. of the ebTEL 2012 workshop co-located with PAAMS 2012. Soft Computing, Springer (2012)
4. Alrifai, M., de la Prieta, F., Di Mascio, T., Gennari, R., Melonio, A., Vittorini, P.: The Learners' User Classes in the TERENCE Adaptive Learning System. In: Proc. of ICALT 2012. IEEE Press (2012)
5. Alrifai, P.: Deliverable 2.1: Conceptual Model's Specification. Tech. Rep. D2.1, TERENCE project (2011)
6. Brusilovsky, P.: Adaptive Hypermedia: From Intelligent Tutoring Systems to Web-Based Education. In: Lecture Notes In Computer Science. 5th international Conference on Intelligent Tutoring Systems (2000)
7. Brusilovsky, P., Karagiannidis, C., Sampson, D.: Layered evaluation of adaptive learning systems. International Journal of Continuing Engineering Education and Lifelong Learning (14), 402–421 (2004)
8. Cain, K., Oakhill, J.V., Barnes, M.A., Bryant, P.E.: Comprehension Skill Inference Making Ability and their Relation to Knowledge. Memory and Cognition 29, 850–859 (2001)
9. Cofini, V., Di Giacomo, D., Di Mascio, T., Necozione, S., Vittorini, P.: Evaluation plan of terence: when the user-centred design meets the evidence-based approach. In: Vittorini, P., Gennari, R., Marenzi, I., de la Prieta, F., Corchado, J.M. (eds.) Proc. of the ebTEL 2012 workshop co-located with PAAMS 2012. Soft Computing, Springer (2012)
10. De la Prieta, F., Di Mascio, T., Gennari, R., Marenzi, I., Vittorini, P.: Playing for improving the reading comprehension skills of primary school poor comprehenders. In: Proc. of the PDSG 2012 workshop. CEUR-WS (2012)
11. Di Mascio, T., Gennari, R., Melonio, A., Vittorini, P.: The user classes building process in a tel project. In: Vittorini, P., Gennari, R., Marenzi, I., de la Prieta, F., Corchado, J.M. (eds.) Proc. of the ebTEL 2012 workshop co-located with PAAMS 2012. Soft Computing, Springer (2012)
12. Gennari, R.: Deliverable 4.1: State of the Art and Design of Novel Intelligent Feedback. Tech. rep., TERENCE project (2011)
13. Gennari, R.: Deliverable 4.2: Automated Reasoning Module. Tech. Rep. D4.2, TERENCE project (2012)
14. Lyon, G., Fletcher, J., Barnes, M.: Child Psychopathology, chap. Learning Disabilities. NY: The Guilford Press (2003)
15. Marschark, M., Sapere, P., Convertino, C., Mayer, C., W.L., Sarchet, T.: Are Deaf Students' Reading Challenges Really About Reading? (In Press)
16. Moens, S.: Deliverable 3.1: State of the Art and Design of Novel Annotation Languages and Technologies. Tech. Rep. D3.1, TERENCE project (2012)
17. Slegers, K., Gennari, R.: Deliverable 1.1: State of the Art of Methods for the User Analysis and Description of Context of Use. Tech. Rep. D1.1, TERENCE project (2011)
18. TERENCE Consortium: TERENCE web site, http://www.terenceproject.eu
19. Vaadin Ltd: Vaadin framework, https://vaadin.com/home. Last accessed: July 30, 2012.