Univ Access Inf Soc (2013) 12:369–386
DOI 10.1007/s10209-012-0283-y

LONG PAPER

Stories and signs in an e-learning environment for deaf people

Paolo Bottoni · Fabrizio Borgia · Daniel Buccarella ·
Daniele Capuano · Maria De Marsico ·
Anna Labella

Published online: 5 September 2012
© Springer-Verlag 2012

Abstract An important field for model-driven development of interfaces is the consideration of users with disabilities. Interface design for deaf people presents specific problems, since it needs to be based on visual communication, incorporating unusual forms of interaction, in particular gesture-based ones. Standard solutions for model-driven development of visual interfaces lack specific constructs for structuring these more sophisticated forms of interaction. This paper discusses such issues in the context of the development of a deaf-centered e-learning environment. Sign Languages enter this context as a suitable alternative communication code, both in video form and through one of their most successful written forms, namely SignWriting.

Keywords Deaf-centered e-learning environment · Storytelling · MOF · UIDL · UsiXML · Sign languages · SignWriting

P. Bottoni (✉) · F. Borgia · D. Buccarella · D. Capuano ·
M. De Marsico · A. Labella
Computer Science Department, "Sapienza" University of Rome,
Via Salaria 113, 00198 Rome, Italy
e-mail: bottoni@di.uniroma1.it

F. Borgia
e-mail: borgia@di.uniroma1.it

D. Buccarella
e-mail: danielbuccarella@yahoo.it

D. Capuano
e-mail: capuano@di.uniroma1.it

M. De Marsico
e-mail: demarsico@di.uniroma1.it

A. Labella
e-mail: labella@di.uniroma1.it

1 Introduction

Content accessibility for deaf users is a hidden problem even in the Web era; especially in the past, it was often thought that a written transcription of audio contents was enough for deaf people to grasp information [7]. In 2006, the World Health Organization testified the existence of 278 million people worldwide who were deaf or had hearing difficulties of different severity. This is a crucial issue since deaf people experience many difficulties in communicating with hearing ones. Moreover, the use of national sign languages (SLs), composed of codified combinations of gestures and expressions involving the upper part of the body, is becoming diffuse among relevant segments of the deaf population, including subjects who know a verbal language. The perceptual basis of such languages allows the expression of one's own thoughts in ways that are dramatically different from written/spoken languages related to a phonetic experience which deaf people lack. It has been observed that deafness as a condition hinders the acquisition of written language skills [25], which raises important issues with respect to interfaces for e-learning [1]. This is a concrete deepening of the digital divide often suffered by deaf people. However, the sole inclusion of sign language (SL) videos in Web pages to translate textual contents is not a complete solution, since significant portions of the deaf community do not learn their national SL and prefer to communicate through verbal language, due to social constraints and prejudice. Moreover, systematic inclusion of video clips may be a huge task and is not adequate for a fluent fruition of contents if a high-bandwidth connection is not available. On the other hand, the lack of a universally accepted written form for SLs still limits their use. Hence, in order for e-learning to cover this literacy gap, creative ways of presenting and coordinating interactive visual materials are needed, with novel supports for content comprehension.

The work reported in this paper has been conducted in the context of a research effort aiming at the development of a deaf-centered e-learning environment (DELE), targeting users who are adult deaf people attending university. One of the foundational principles of DELE is to avoid text, whenever possible and not explicitly required by the learning activity, and to opt for a visual presentation of information. DELE design is based on conceptual metaphors, according to the embodied cognition paradigm ([18, 20]), as well as on storytelling ([5, 22]). The design process has especially investigated how metaphors based on the interaction between humans and their environment (e.g., container, path) can facilitate learning. The central metaphor exploited in DELE is the university campus. Users can browse the campus environment via personal avatars, exploiting intuitive body-based actions. Moreover, typical concepts from Web sites and social networks (e.g., personal pages, forums) are mapped to domain-specific entities, and an exhaustive mapping links concepts typical of the learning experience (and specifically of e-learning) to these entities (e.g., the campus main square represents the forum, personal houses stand for users' personal pages).

All the learning processes supported by DELE are implemented as (possibly interconnected) stories. Each story is defined as a path from a starting place to a conclusion, through several steps and detours. The story progress is visually rendered by an avatar that moves along the corresponding path, and textual information is omitted unless it is part of the intended educational contents.

Creating educational material and processes according to the DELE conceptual structure requires a suitable authoring support. To this purpose, DELE integrates a StoryEditor component allowing tutors with no specific technical experience to design learning paths as stories, by connecting several nodes of different types. The choice of a node triggers the automatic generation of object code for Web pages. The implementation of a learning process relies on the mapping of stories onto navigation paths.

Navigational, graphical and activity structures in the story pages are specified via the UsiXML User Interface Description Language (UIDL) [21], which provides several models for the expression of the different aspects of DELE design, namely task coordination, flexible style of presentation, as well as typical e-learning. In addition, a metamodel for the domain-specific language of stories is expressed in the form of a conceptual class diagram. The types in the metamodel are associated with specific patterns, relating the definition of stories as paths to the concrete organization of the interface. The implementation of the StoryEditor therefore corresponds to a reification of these patterns into specific procedures for the construction of specific instances of a story. In a similar way, different types of stories are associated with specific forms in which user interaction and content presentation can occur.

Specific actions have been defined to support the specification of gesture-based forms of interaction, which are not currently supported by UsiXML. In particular, in order to use sign languages (SLs) for content presentation in DELE, writing facilities are introduced based on SignWriting[1] (SW), using a novel specification of an abstract syntax for SW. This is at the basis of the design of the SignWriting improved fast transcriber (SWift) graphical editor. With SWift, a user composes SW sentences using visual symbols (glyphs) to represent SL configurations, movements and facial expressions.

This paper is organized as follows: After discussing related work, the use of UsiXML for the formal description of fully iconic interactive contents is briefly sketched. Then, the paper illustrates the use of StoryEditor to produce Web pages from abstract models and maps such models onto the suitable metamodels provided by UsiXML. Finally, the SW representation of signs and gestures is summarized, proposing a corresponding metamodel.

2 Related work

Storytelling is the basic abstraction underlying the definition of storyboards in user interface development; the integration of the latter with model-driven approaches is discussed in [17]. However, interface development is often treated as a minor issue within projects related to storytelling and e-learning settings for deaf people. Moreover, it is to notice that, as in many advanced educational experimentations, children rather than adult learners are chosen as target users.

Attempts to address general accessibility issues have been made by the World Wide Web Consortium (W3C), with the Web Content Accessibility Guidelines (WCAG) document. However, there is scarce comprehension of problems encountered by deaf people, and the most addressed disability is blindness, while a written transcription of audio content is considered enough to support deaf users. WCAG1.0 guidelines (1999) deal mostly with problems related to labeling and transcription of audio content, leaving out alternatives related to SLs. WCAG2.0 (2008) better addresses such issues. For instance, the success criterion 1.2.6, which is a prerequisite to get the highest level of compliance (AAA), states that “Sign language interpretation is provided for all prerecorded audio content in the form of synchronous media types.”

[1] http://www.signwriting.org/.

The Signed Stories project[2] is addressed to primary school. The objective is to make children's stories accessible in British Sign Language (BSL) and also use animations, pictures, text and sound (mostly intended as vibrations, of course) to improve the literacy of deaf children. The MOODLE Course Management System[3] has been adapted at the University of Bristol Centre for Deaf Studies[4] (CDS) to deliver in BSL e-learning contents otherwise available in written English. The Digital Storytelling Program at Ohio State University[5] proposes showcases, presentations, publications and workshops where deaf and hearing participants learn to use digital tools and interactive story circles to craft narratives.

Little has been done as to model-driven development of interfaces for deaf people. Present approaches to such design activity mostly focus on assistive interfaces to support blind people, with extensive use of the alternative audio channel ([16, 33]). On the other hand, gesture-based interaction has been prominently seen as an integration of speech (e.g., see [27]). Work on the Model-based lAnguage foR Interactive Applications (MARIA) project is currently aimed at integrating support for gesture-based interaction, but its model of gestures is typically related to typical movements and movement sensors as included in games or multitouch interfaces ([24, 29]).

Efforts to provide specifications of SLs are currently mostly focused on the generation of movements of avatars [8]. The Hamburg Notation System is the basis for Signing Gesture Mark-up Language (SiGML) [12]. However, this follows the now outdated notion of gestures as mainly defined by hand and arm motions. On the contrary, it is nowadays accepted that information is conveyed in a more structured multilinear way, through gaze, hands, facial expression, head and shoulders. This is also the reason why it is unfeasible to directly transcribe signs from the written form of a verbal language. On the other hand, the whole set of elements which characterize SLs can be found in the definition of SignWriting by the former choreographer Valerie Sutton [31].

As a remark about modalities of general-purpose information communication for deaf users on the Web, it is interesting to consider some reflections from Fajardo et al. [13]. At present, regional as well as national variations of sign language form a group of under-represented language minorities in the digital world. Thus, members of the deaf community are usually confronted with Web sites where both information and navigation compasses are expressed in non-native language and that do not include their preferred tongue. This may, and very often does, cause severe accessibility barriers. The way to ensure the social integration of the deaf community is to properly incorporate sign languages into the information and communication technology (ICT). Along these lines, a solution that is often exploited on the Web, and that is proposed for e-learning as well, is to integrate verbal language information with videos where a person using SL expresses it in an alternative form. This strategy is often included in more sophisticated approaches that are designed specifically for the Web, to create whole sites by exclusively using SL for both content and navigation paths, the latter being the trickiest to render. With the hypervideo technology, links are inserted in a video so that the user can retrieve further information about the concepts conveyed in some cut-scenes (for a review see [10]). The links are represented by text or by static images, but these cannot properly convey concepts as they were expressed in SL.

Based on this technology, Fels et al. [14] developed the Signlinking system. Each Signlink is an author-defined time slice within the video clip. When the video reaches a Signlink, a link indicator is displayed. The user can choose whether to follow the link or continue playing the content. Though esthetically and technically attractive, such solutions present two main drawbacks. First, it is often unfeasible to set up the recording of a high number of clips to transmit the complete information. Second, video fruition on the Web is often hindered by bandwidth limitations, which waste developers' work and frustrate users. In addition, lack of standardization makes things even more complex. A recent structured effort toward deaf inclusion in distance education is the Dicta-Sign project. Dicta-Sign[6] is a 3-year consortium research project. It aims at allowing the use of sign language in various human–computer interaction scenarios [11]. Among the addressed research and design topics, recognition and synthesis engines for signed languages are worth mentioning. In this context, Dicta-Sign aims at combining work from the fields of sign language recognition, sign language animation via avatars, sign language linguistics and machine translation, with the goal of allowing deaf users to make, edit and review avatar-based sign language contributions online, similar to the way people nowadays make text-based contributions on the Web. The use of avatars animated by a knowledge-based sign synthesis engine is one of the main features in the approach [15]. However, according to the deaf researchers involved in the VISEL project, the quality of avatar animation is still too far from being fully satisfying. This opinion, which was one of the recurring topics of our joint work, is confirmed by the work of other researchers [19], where there is strong evidence for involving deaf

[2] http://www.signedstories.com.

[3] http://moodle.org.

[4] http://www.bris.ac.uk/deaf.

[5] http://digitalstory.osu.edu.

[6] http://www.dictasign.eu.

stakeholders in all steps of design and implementation of deaf-oriented tools. An appropriate structuring of contents as well as a way of transcribing text in a SL-like form seems to be at present the most convincing strategy.

Among the technologies on which StoryEditor is based, it is important to mention the WiringEditor component of WireIt,[7] an open-source JavaScript library to create Web interfaces for dataflow applications, visual programming languages, graphical modeling or graph editors. In turn, WiringEditor exploits JavaScript Object Notation (JSON),[8] a lightweight data-interchange format, which is becoming a de facto standard for data exchange in AJAX applications. Via the JSON Language Definition one defines a visual language in terms of modules and WiringEditor options. WireIt is based on the Yahoo! User Interface (YUI) library,[9] a set of CSS and JavaScript utilities and controls for building interactive Web applications, and on inputEx, an open-source JavaScript framework to build fields and forms for Web applications.[10]

3 Telling iconic stories

The adopted approach to the development of an e-learning environment designed in a fully visual fashion relies on storytelling. Indeed, the visual modality is often a central part of storytelling, especially when children are involved [32]. It has been observed that stories activate visual thinking,[11] and from this point of view, they are a tool of paramount importance to address learners who mainly adopt such cognitive strategies. This does not hold only for deaf people. On the contrary, stories represent one of the main tools by which people mentally make up a synthesis of experienced life, especially before a symbolic level of understanding is fully acquired [5]. Although this mechanism is partially overcome by symbolization in adults, deaf people maintain a deep connection between symbolic and iconic levels of meaning, due to their visual approach to knowledge. In other words, deaf people are essentially visual [1]. In the development of storytelling environments, two (non hierarchical) description levels must be considered:

1. A *diachronic* level, representing the action flow along a story-path from the starting to the end place.
2. An *iconic* level, determining appearance and influencing the “felt quality” [17] of the user experience.

Modeling tools able to describe both levels are needed to allow tutors to design and run iconic stories from scratch. This is the basis for designing the StoryEditor for DELE.

4 UIDLs and UsiXML

UIDLs allow the description of UIs in an implementation-independent way. Through them, it is possible to specify the UI features at several levels of abstraction. These levels are described in the *Cameleon* reference framework [6]:

- *Tasks and Concepts*: describes user tasks that have to be realized, in a way independent from interaction modalities and concrete UI elements.
- *Abstract UI* (AUI): provides a modality-independent description and represents a UI as a collection of abstract containers and components.
- *Concrete UI* (CUI): includes modality-specific elements, but without reference to specific platforms or implementations.
- *Final UI* (FUI): describes the UI as perceived by users in specific hardware/software platforms.

UsiXML implements the Cameleon framework via different models for different levels of abstraction (with the exception of the final UI, which needs to be described directly in the target platform and is not specified in UsiXML). Model-to-model transformations allow designers to consistently move from the *Task Model* to the *CUI Model*, through the *AUI Model*. These are defined in a homogeneous formalism based on transformation rules, typically in the form of graph transformations.

In UsiXML *Task Model*, relationships between tasks are specified via the ConcurTaskTree [23] and LOTOS [3] formalisms. In the *AUI Model*, a UI is represented using *Abstract Individual Components* (AICs) and *Abstract Containers* (ACs), together with their relationships. The *CUI Model* describes graphical and vocal modalities of interaction with interface elements and the associated behaviors.

The abstract models—together with abstract-to-concrete transformation rules—represent the information needed by the environment to determine the input parameters for an object code generation process. However, lacking a model for the final UI, a more direct implementation strategy has been adopted, though remaining bound to the Cameleon framework and leveraging UsiXML to provide specifications for model-based generation activities within StoryEditor. Patterns, rather than transformations, are used to specify the relations between tasks, abstract UI and concrete UI, while specific constructs define transformations from concrete UI to final UI.

[7] http://neyric.github.com/wireit.

[8] http://www.json.org.

[9] http://developer.yahoo.com/yui.

[10] http://neyric.github.com/inputex.

[11] http://www.learningandteaching.info/learning/dalebruner.

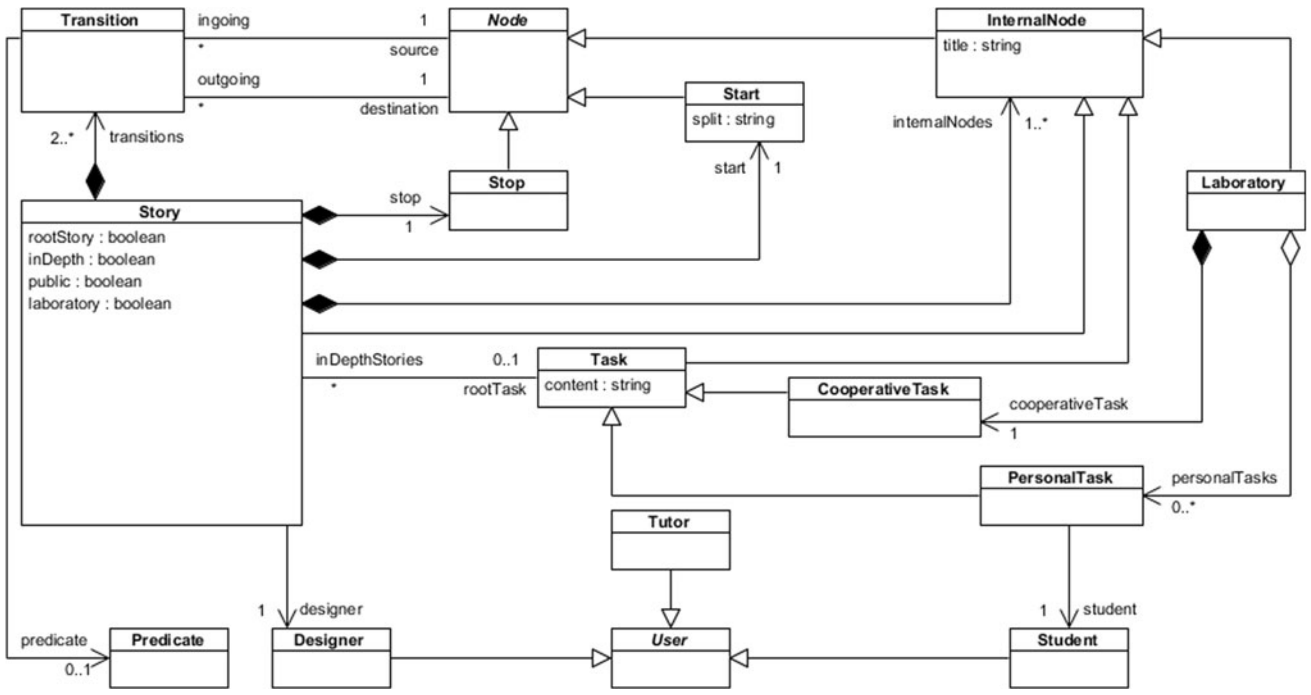


Fig. 1 The formal metamodel for story-paths

5 The StoryEditor

The discussion in this section starts from the diachronic description level. From a computational point of view, stories can be seen as workflows: (learning) processes within an organization (e.g., university), in which tasks are assigned to actors (e.g., students and tutors). The types of processes and activities devised for the specification of stories are described in the metamodel of Fig. 1, using MOF[12] (OMG's Meta Object Facility) syntax, where each class is an instance of the *Class* metaclass. The abstract class *Node* defines the basic elements that can be connected to build a path, represented by the *Story* class. A story contains a *Start* node, a *Stop* node and a set of *InternalNode*s, with arbitrary links among them. A *Story* can have *sub-stories* since an *InternalNode* can be a *Story* in turn. A *Transition* represents a connection between two nodes within the same sub-story. Every node can have any number of ingoing and outgoing transitions.

In order to manage the types of workflow derivable from the DELE metamodel, the YAWL[13] (Yet Another Workflow Language) workflow management system was first tested, which provides an editor for workflow description and a workflow engine for story-paths execution. While initially this seemed sufficient to provide the story authoring environment for DELE, it finally emerged that it could not comply with the major requirement for DELE to have a full and easy integration of the story editor as a browser-based application. In particular, it is not possible to easily customize and represent iconically those pages which represent "crossroads", that is, pages where the student chooses the next node to visit among several equivalent possibilities. As a matter of fact, YAWL allows easy customization only of the pages to view/edit the contents of the task. A further limit is the inability to display the complete map of a path, which is a core prerequisite for the execution environment, as this allows students to know where they stand with respect to the overall learning task and to move arbitrarily between active nodes. Other issues were the inability to jump within a path and the impossibility, in a linear topology, of going back rather than forward. Finally, preservation of context across different stories required particular solutions. As a result, it was decided to develop a specialized visual editor for the domain-specific language of stories according to the metaphor adopted in DELE. In particular, for each class of the metamodel, specific procedures were defined for their instantiation and for the construction of the Web pages offered to the interaction of the DELE users.

The implementation of StoryEditor was therefore based on WireIt, extending its WiringEditor component to support the story-paths visual language, while inheriting from it some of the common features of most visual editors. The following subsections give a detailed explanation of the three steps in the definition of interfaces for stories.

[12] http://www.omg.org/mof.

[13] http://www.yawlfoundation.org.

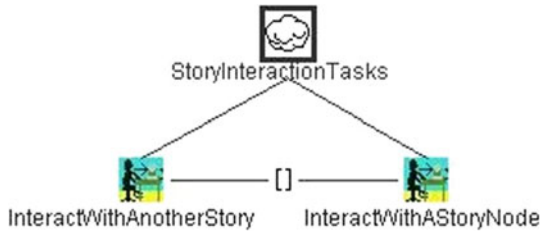


Fig. 2 Task Model for *StoryContainer*

5.1 First step: visual language definition

5.1.1 *Story structure definition*

While arbitrary topological structures must be designable by educational tutors, it is also necessary to avoid sources of ambiguity in the final result, as well as to make automated syntax check possible. For this reason, the formal metamodel of Fig. 1 provided the basis for the definition of admissible learning paths.

In particular, according to the metamodel, a node along a path can be either a *Task* (where to perform a proper learning activity is performed) or a story in turn (i.e., a sub-story). There is no theoretical limit to the nesting of stories. As shown in Fig. 2, from a task node, a student can also access in-depth sub-stories. The latter may not belong to the normal path of the main story, but may create entirely alternative paths, which can be followed by the student within the navigation.

An important extension of the internal nodes class is the *Laboratory* node. *Tasks* can be further divided into *PersonalTasks* and *CooperativeTasks*. In particular, in a *Laboratory*, students perform activities articulated in a number of *PersonalTask*s and a single *CooperativeTask*. Each student has assigned a personal task to be performed asynchronously in isolation, while cooperative tasks can be performed in different ways (e.g., through shared documents) and require the (synchronous) presence of all participants. *Active* nodes are defined as those that can be immediately affected by a transition, for example, the next one in a sequential path. Transitions within sub-stories are constrained by a number of patterns associated with the metamodel. When accessing a sub-story, and as far as unvisited nodes are available, the student can choose to visit one of those active. For example, when entering a linear sub-story, the only active node is its starting one. When a sub-story terminates, that is, all of its nodes have been visited, the student returns to the parent story (if it exists) and the context of the latter is restored.

5.1.2 *Managing content*

While navigating across stories, a notion of context emerges, through which a student maintains conceptual orientation, through reference to a parent story from within a sub-story. A designer can specify different ways in which to represent the current context of an activity, via context-sensitive properties, for example, background color, font style or page layout or more specific content-related ones. StoryEditor currently manages three types of “context”, intended as a set of characterizing elements, for example, a specific font or a tag to index the content. These elements can be defined according to the type of a node:

1. Each text node allows connecting insights to the learning material (text/images), via the insertion of a set of tags (simple text strings) for each content element chosen by the designer.
2. Each network node of type “Insight” defines its own set of tags.
3. Each node of type Network or Laboratory defines its own set of variables that can be boolean, integer or string.

For types 1 and 2, during code generation, each time that a text node defines a set of insights (hence, a set of tags associated with each insight), the system proceeds as follows: for each piece of text/image related to an insight, all the tags defined for it are read; afterward, all Insight networks that define one or more of these tags are searched in the DB. These networks are retrieved from the DB if they already exist or are generated on the fly if new collections of tags are defined. Finally, a link to each such network is connected to the initial text/image. Moreover, the networks are generated as “BOXES”, each further characterized by its own type. The type of the box defines a fixed look and feel as a “frame” for its contents. Within a text node, clicking on a portion of text/image a page opens showing all the boxes for all the insights associated with it, if any exists. Clicking on a box, the user can navigate the related network, whose graphic will be displayed within the frame of the box.

As for the third type of context, the “variables” of nodes of type Network or Laboratory can be used to transfer information between network nodes, to define conditions that would allow the implementation of XOR splits type, or handling of additional help materials provided for each node in the generated path-history generated.

5.1.3 *Task content abstract description*

Task node internals are described by UsiXML models, to consistently define the iconic level of story description. A few main categories have been identified for DELE story pages (e.g., “story container” or “star story access”), devising a uniform structure for each of them in the form of abstract descriptions, specified in terms of *page patterns*. These patterns rely on instances of the above-mentioned

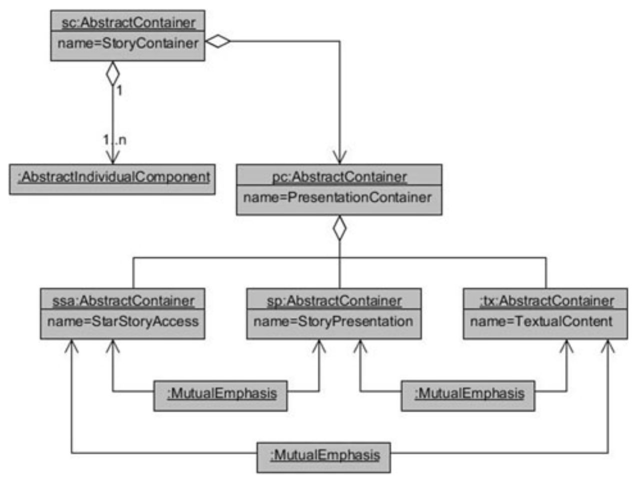


Fig. 3 The page pattern for *StoryContainer*

UsiXML's Abstract Interaction Objects (AIOs)—either AIC or AC—and describe the structure of all pages in a category. To develop the models needed for a category, a Task Model is first provided. Then, the abstract page pattern is shown, and a first transformation is provided between tasks and appropriate AIOs. Task Models are specified in the IdealXML[14] editing environment using the LOTOS syntax for temporal operators. A main common structure for all pages within DELE is defined, called *StoryContainer*. In a container, the designer can define the main actions that can be performed by users within a story, while the reification process outputs adequate graphical structures for such actions. The Task Model for *StoryContainer* is shown in Fig. 2, where the two main task patterns are presented:

1. *InteractWithAnotherStory:* the user can enter a story by leaving the current story.
2. *InteractWithAStoryNode*: the user can enter a node from within the current story.

In Fig. 4, both "abstract" (*StoryInteractionTasks*) and "concrete" interaction tasks (e.g., *InteractWithAStoryNode*) are shown via IdealXML standard icons. The *alternate choice* operator "[]" expresses the relationship between interaction tasks as the possibility to choose only one of them at each time.

The page pattern for *StoryContainer* is given in Fig. 3 in terms of instances of AIC and AC classes from the UsiXML AUI model and of relationships between them.

A main *PresentationContainer* (an AC) and several AIC instances form the pattern. The AC is composed of several different "page contents" related by *MutualEmphasis* (i.e., they cannot be shown together). The AUI Model derives from the Task Model, where either an AC or an AIC is

[14] http://www.usixml.org/index.php?mod=pages&id=15.

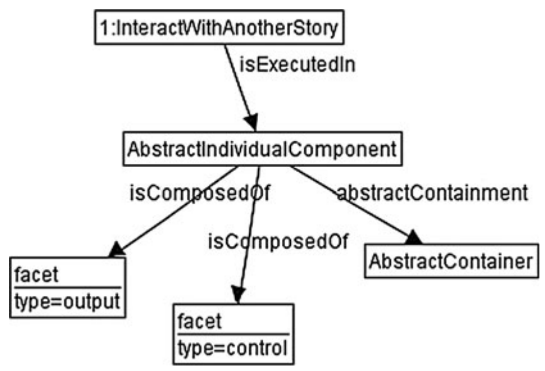


Fig. 4 A pattern relating a type of story with its AUI Model

associated with each task. The *StoryContainer* AUI Model is defined in terms of specific patterns leading to suitable organizations of the AICs, to be then reified into concrete components in order to present the content of a learning task and steer the associated interaction.

Figure 4 shows a pattern which specifies how to express the *InteractWithAnotherStory* task. As in [30], *InterModelRelationship* instances are represented as associations between model components. In particular, a task is executed within an AIC, which is contained in an AC and is composed of two facets: a *control* allows the AIC to activate a visualization into another component in the concrete UI, and an *output* allows reification of the AIC as an iconic image component. In a similar way, the CUI Model is obtained from the AUI Model. Each AIC in *StoryContainer* is implemented as an *ImageComponent*. Page dynamics is defined by specifying the activities to be associated with interesting events. For example, in Fig. 5, a *click* event causes a *graphicalTransition* to start on the connected *graphicalContainer*. Similarly, a *mouseOver* event produces a graphical transformation in the AIC itself.

As another example, the *StarStoryAccess* structure represents the access door to "star" stories, where multiple paths can be entered by the user without an imposed order. The Task Model of such structures defines only two possible task patterns: users can interact with a story-path or find information about a path (Fig. 6).

The page pattern of *StarStoryAccess* extends the diagram of Fig. 3, detailing the *StarStoryAccess* tree (Fig. 7). The structure of *StarStoryAccess* simply states that a star story must have at least two paths, which can be independently explored. In fact, if only one path can be followed, a "linear" path would result.

Both task patterns described for this UI have to be executed in a single AIC instance; thus, the AUI model has to provide a multifaceted graphical interaction component.

Each task is associated with one of the two facets of the GIC: in Fig. 8, it is shown that the *InteractWithAStoryPath* task is responsible for the *navigation* facet since the AIC

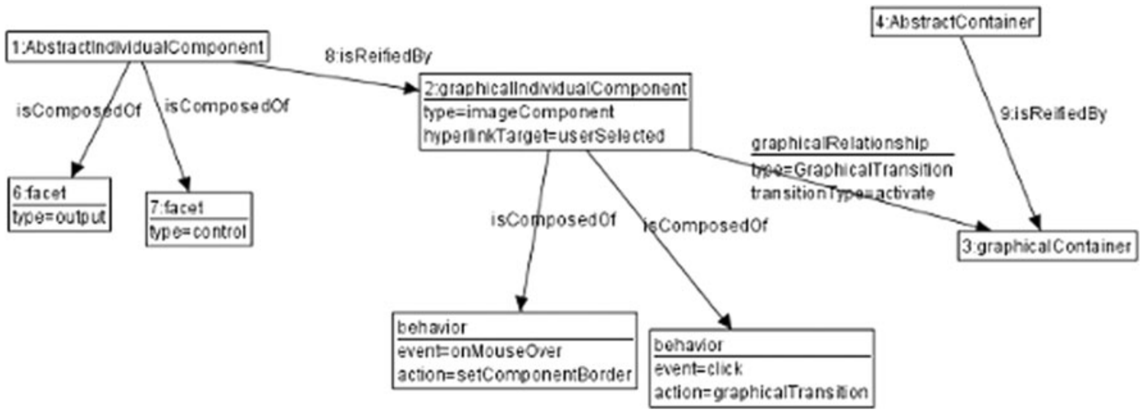


Fig. 5 A pattern for the AUI-to-CUI reification of *StoryContainer*

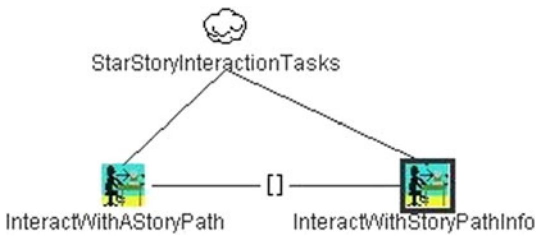


Fig. 6 Task Model for *StarStoryAccess*

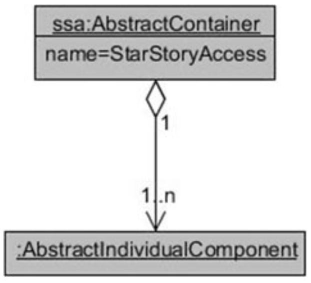


Fig. 7 The page pattern for *StarStoryAccess*

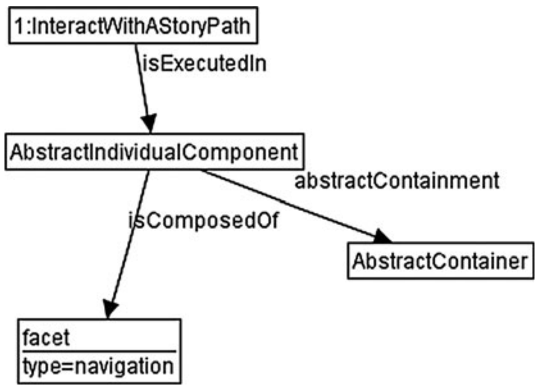


Fig. 8 The pattern for InteractWithAStoryPath

itself will work as a navigation target when it is activated. On the other hand, the *InteractWithStoryPathInfo* task requires an *output* facet for the AIC since a textual label will be shown to the user as a *mouseOver* event is performed. Again, associated behaviors can be specified, for example, when a *click* event is triggered, the GIC has to display the page contents, expanding to full page size through an animation. In a similar fashion, the *mouseOver* event causes the GIC to react by visualizing information about the path it represents (i.e., a graphical output text is put near the image component).

The set of models and patterns provide the input to StoryEditor to calculate the generation parameters: for example, as the page pattern shown in Fig. 5 states that multiple AICs instances could be put in a *StoryContainer*, StoryEditor should ask the story designer to indicate the exact number of these elements; similarly, since the AUI-to-CUI relationship patterns specify that AICs are to be reified as *ImageComponent* elements, the designer will be asked to provide these images, etc. After all the needed models have been provided, StoryEditor is ready to run. Once launched, StoryEditor reads the story content definition and structure from a JSON Language Definition module.

5.2 Second step: story-paths editing

In the next step, the UsiXML descriptions are translated into JSON as well. Visual language editing only requires editing the resulting definition files. All the changes that are made to the structure or content metamodels are immediately reflected in the visual editor. The view resulting from the above specifications is shown in Fig. 9.

The editing window provides a list of available modules and tools on the left side of its GUI layout. A node of a given type is created within the story by inserting it into the diagram in the main drawing area, in the center of the window, via drag and drop of the appropriate module. A number of communication modules have been developed to allow the active participation of students in activities

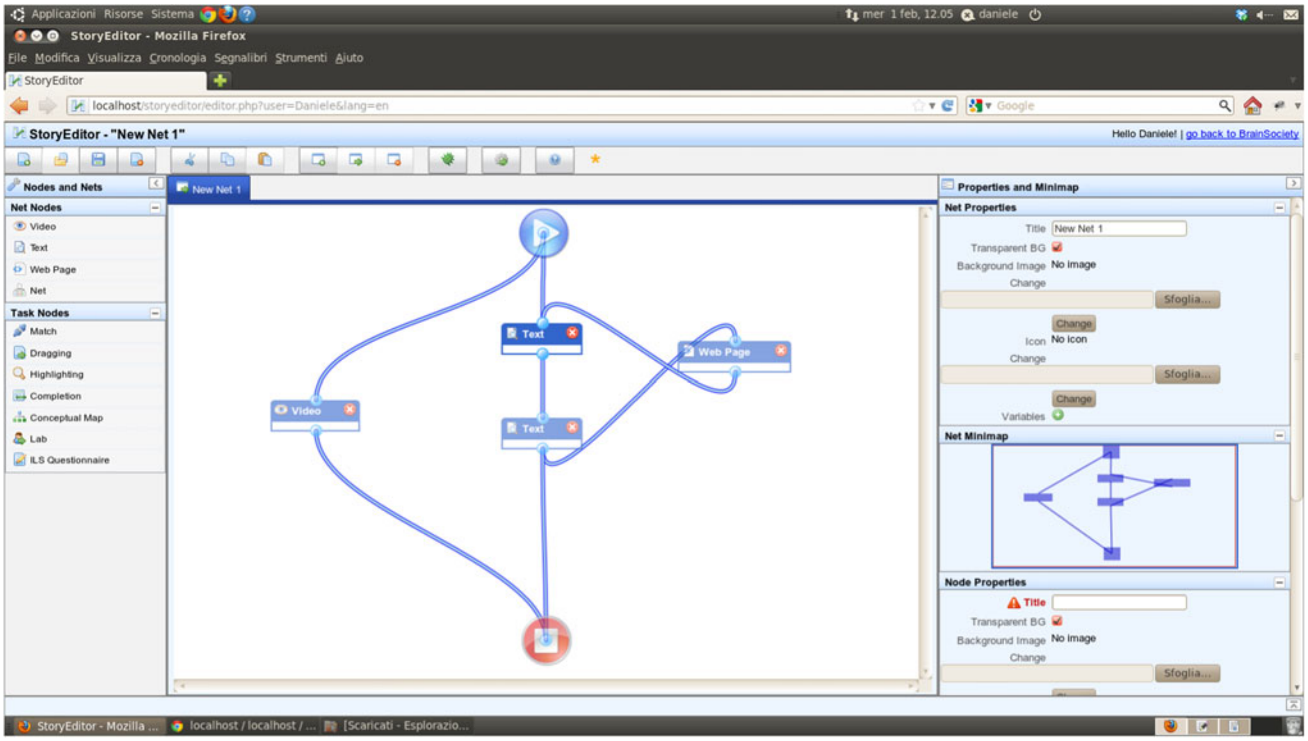


Fig. 9 A screenshot of StoryEditor

supported by the platform, through forums, private messages, chat and annotations within text. For each of these modules, functions for the insertion of the content have been implemented: these contents can be produced by the student by composing contributions expressed in different codes: currently, only written and/or video are supported, but the inclusion of an editor for SignWriting is under way. A video recorder and player have been directly integrated within DELE, so that students do not have to use external applications. To this end, DELE has been interfaced with the Flash streaming server "Red5", which allows real-time acquisition of video streams and their recording on the server.

As for the educational content of DELE, a library has been implemented to create and enact contextual insights, which are made accessible by clicking on particular portions of text/image selected during the design phase of the learning path. These insights (called "boxes") are categorized on the basis of a list of predefined types, each of which defines a main content and a particular graphic aspect. Clicking on a box, the student accesses sub-stories for better insight, within the same graphic frame as that of the box, and which can be displayed next to the text or full screen.

By establishing a precise set of characteristics for each node type, StoryEditor asks the designer to provide all the information required for the effective generation of the designed teaching stories. In particular, input forms are generated by the StoryEditor to ask the designer to insert the predefined features of each node in the model (which are instances of types in the metamodel). These forms follow a schematization of the transformations from abstract UI to concrete UI: the output JSON description of each node provided by the editor is just a "translation" in JSON of the concrete UI in UsiXML. The engine for the code generation of the stories is implemented starting from the JSON description provided as output by the StoryEditor. In this way, when a new story defined through the StoryEditor is published, it invokes a script that, given the description of this input story, parses it and generates the necessary modules in DELE and, finally, returns the URL of the story generated. At this point, the editor opens that URL to run the story. The generation is currently available for all types of nodes, including activities and workshops. Finally, the runtime environment of the stories generated has been implemented so as to define generic libraries for the execution of each story and to view and manage the reference map for navigation.

All the types of nodes that can be inserted into a model of history have been associated with the definition of possible content that can be handled through the StoryEditor. The properties of the "Home of a story" have been implemented as the properties of a network; therefore, there is no specific real node to define them. The node "Presentation page" has taken the name of "Video", because its content is a video featuring LIS and can be entered multiple times within a network of a model and not necessarily at the beginning. The node "Job" was divided into its two possible instantiation types: "Text" and "Web page". It was decided to define a different node for each type of content in order to enable the designer to immediately identify, thanks to the icon of the node, the type of the contents of the latter. The "Activity Nodes", through which a student can carry out a practical activity, are classified as follows: "Pairing", "Drag", "Highlight", "Concept Map", "Completion", "Laboratory" and "ILS Questionnaire."

Through StoryEditor, it is now possible to define the contents of all the Activity Nodes, the structure of these nodes being completely generated by the script for the code generation of the stories, managing all the auxiliary information needed to execute them as well. For example, since the Laboratory activities implementation is based on the OpenMeetings open-source package,[15] all the OpenMeetings-specific data are instantiated in the code generation phase for such kind of nodes. Moreover, for each new generated node, a new node-specific forum is created in order to host all the discussions related to such node. Both textual portions and images included into texts can be connected to insights through the mechanism of tags: the designer assigns a set of tags to the selected resource (portion of text or image) and a set of tags to a network of insights; the script for code generation later realizes the connections between resources and insights that declare the same tags. It is planned to have links to networks of insights defined in different models without having to know their names or their internal structures. The topologies that are provided by DELE are linear and "star". The ability to provide branches of type AND and OR was therefore maintained only on the nodes "Start", while the ramification of XOR has been maintained available on any type of node.

In order to facilitate the tutor in the process of designing a story-path, each time a new sub-story is created, the editor automatically inserts start and stop pseudo-nodes in the diagram. Visualization and navigation within sub-stories are made easier by creating a different window tab for each of them. An in-depth sub-story can be attached to a task node by linking it, for example, to some words of the text contained in the node. Different types of in-depth sub-stories have been defined, according to their educational role. The first type is *Recommended Further Reading*. This includes in-depth contents that are statically created and linked by the course designer in the modeling phase of a story, or even dynamically added by the tutors, for example, to address a specific students' difficulty in a particular activity. A second type is *Past Correlated activities*. They refer to past activities already performed by the student and allow repetition and reworking of the concepts learned. Finally, *Personal Insights* are defined. These insights are added by the students themselves on the basis of their personal interests [4]. A syntax check verifies the story correctness and completeness based on the metamodel descriptions, which are associated with sets of constraints that a story must comply with to be published. Finally, *StoryEditor* uses a JavaScript adapter to provide loading and saving functions. It connects to a MySQL database through AJAX calls to a PHP backend to both store the wirings and retrieve them.

[15] http://code.google.com/p/openmeetings/.

5.3 Third step: story-paths translation

Once the editing phase is concluded and the syntax check has been passed, the new story can be published. The *StoryEditor* saves the story description in JSON, which will be interpreted in the runtime environment. Modules from the story-path visual language are translated in JSON according to their metamodel description.

A "Correction and support" atomic task is automatically inserted after each activity. The choice of the content for this task is left to the tutors, so they can verify how learning activities have been performed. A cycle is eventually generated, in which tutors can support students by inserting links toward in-depth sub-stories relating to their possible errors. Verification is performed differently for laboratories and for simple tasks. As for collaborative activities, tutor verification starts only after all the personal and cooperative tasks have been completed by students.

6 Story engine

For each type in the metamodel, there is a transformation pattern from JSON description to HTML pages (PHP), which exploits CSS specifications. Going in further detail, each type corresponds to a triple <JSON, CLIENT_STRUCT, SERVER_STRUCT>, where the first element is the JSON output description provided by the StoryEditor, and the other two elements are defined in the following way.

CLIENT_STRUCT:

- HTML: page structure
- CSS: visualization features of page elements
- JS_INTERACTION: script defining the behaviors for interacting with page elements
- JS_AJAX: script implementing module-related behaviors for data exchange between client and server

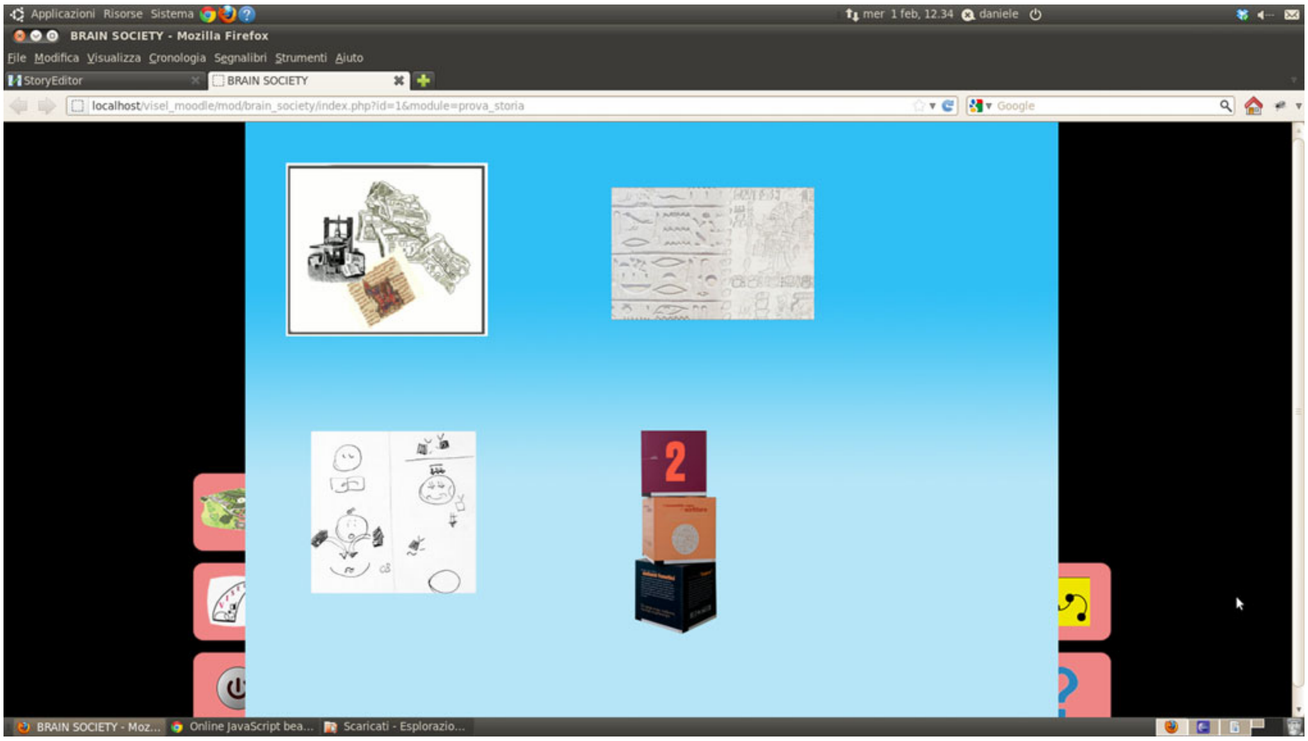

Applicazioni Risorse Sistema
mer 1 feb, 12.34 daniele
BRAIN SOCIETY - Mozilla Firefox
File Modifica Visualizza Cronologia Segnalibri Strumenti Aiuto
StoryEditor
BRAIN SOCIETY
localhost/visel_moodle/mod/brain_society/index.php?id=1&module=prova_storia
Google
2
BRAIN SOCIETY - Moz...
Online JavaScript bea...
Scaricati - Esplorazio...

Fig. 10 Initial page of a “star” story-path

SERVER_STRUCT:

- MODULE: PHP object associated with the module on the server
- CONTENT_MANAGER: PHP object allowing the definition, generation and retrieval of module contents. It takes the content of files in CLIENT_STRUCT, adapts it to the current navigation instance (e.g., user, session) and returns it.

Each type in DELE metamodel is concretely generated, starting from its JSON description, as a DELE module, that is, as a set of 6 files composing the type SERVER_STRUCT and the CLIENT_STRUCT. In the generated DELE runtime environment, students follow paths composed of learning activities and stories within stories. When all nodes within a sub-story have been visited, the latter is marked as completed and the global context of the parent sub-story (if present) is recreated. When navigating along sub-stories, each time a sub-story is entered, and until there are nodes to be accessed, students can choose the next learning unit among “allowed” nodes. In a linear path, for example, the only allowed node is the first node at the beginning of the story. Inside a laboratory, students can freely move between personal and cooperative tasks. The set of students attending the laboratory has to be known to allow synchronization: when one student tries to join the cooperative task, a message is sent to all students not currently attending this task, requesting them to enter it. If all students accept the request, synchronization is reached and the shared work can be done. Otherwise, students are redirected to their personal tasks. According to the order of visit, or to the alternative paths a student must—or choose to—follow as in-depth sub-stories during execution of a story-path, the student “lives” a personal story. Hence, different sets of *Past Correlated Story-Paths* and *Personal Insight Story-Paths*, one for each student, are maintained and shown in a laboratory. The latter is considered concluded when all its tasks (both personal and cooperative) are done, and the final tutor verification is passed.

Other specific custom services are provided in order to execute each DELE node. At runtime, the data within each node instance are selected for viewing and/or updating. When a task is scheduled, the engine will notify each custom service associated with the task that there is an activity ready to be delegated to it. Hence, the custom service performs a checkout of the task data and generates an appropriate editing form based on the CUI description provided for that node by the *StoryEditor*. Two examples of generated final UI are shown in Figs. 10 and 11, presenting two initial pages of “star” story-paths. In particular, the screenshot in Fig. 11 shows the presentation page of a star story-path encountered as a sub-story of the star-path in Fig. 10, with a button to return to the parent node.

Figure 12 shows how a piece of text is linked to a box with a network of insights. The box is displayed on demand when the user clicks on the highlighted text. Figure 13

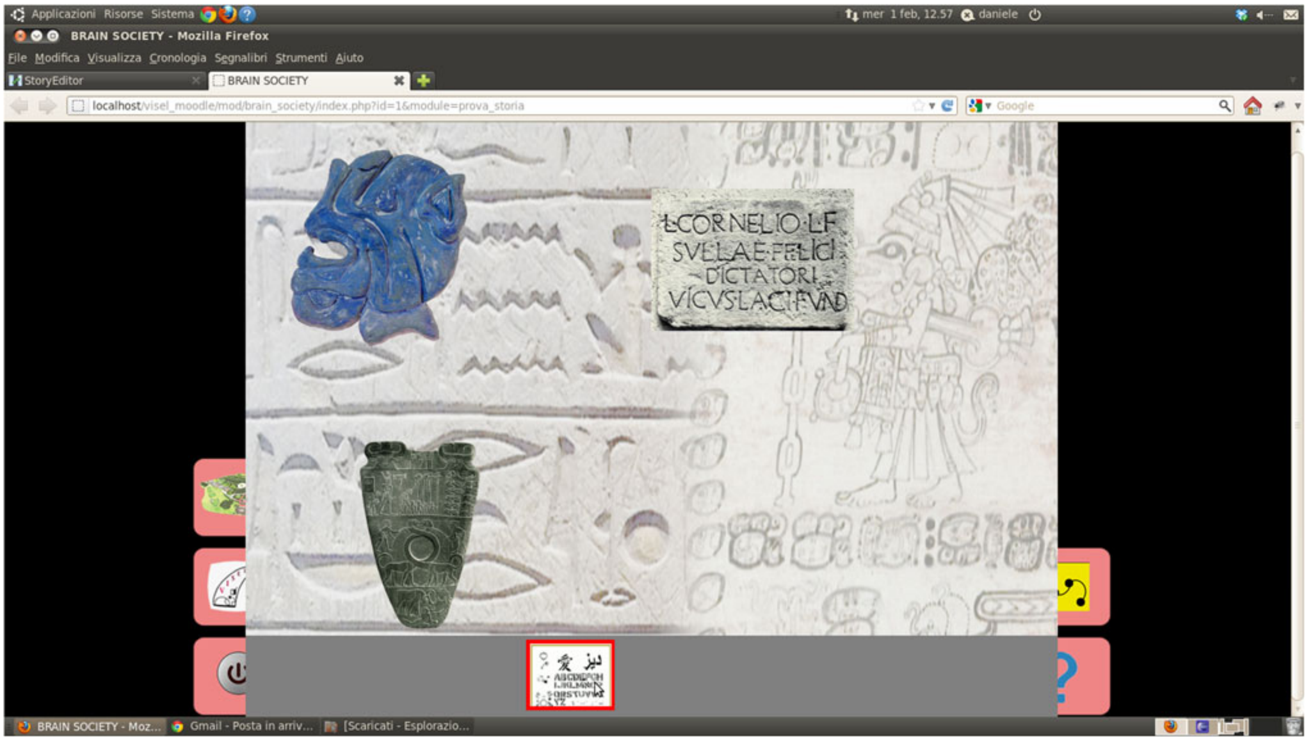

Applicazioni Risorse Sistema
BRAIN SOCIETY - Mozilla Firefox
File Modifica Visualizza Cronologia Segnalibri Strumenti Aiuto
StoryEditor
BRAIN SOCIETY
localhost/visel_moodle/mod/brain_society/index.php?id=1&module=prova_storia
L·CORNELIO·L·F SVLLAE·FELICI DICTATORI VICVS·LACI·FVND

Fig. 11 Initial page of a "star" sub-story, with a button to return to the parent node

shows one of the insights opened (in this case, a video in SL is being played). Since both text highlights and insight boxes can be too intrusive, an icon with a special pair of glasses allows choosing the visualization of possibly present elements of this kind. After finishing, taking the glasses off will return the "clean" page.

Finally, Fig. 14 shows how a student can link a personal comment to a piece of text. The presence of comments is indicated by balloons on a specific icon.

7 SignWriting in DELE

SignWriting[16] (SW) is a system for writing expressions of sign languages. It exploits an alphabet of glyphs, used to produce a transcription of any SL in the world. At present, it is one of the most well-accepted "written" representations of SL within the deaf community. Compared with other notations, SW can express a signed sequence by itself, without any accessory description and/or annotation written in a different language (typically the written form of a verbal language). SW glyphs are all gathered and organized in the International SignWriting Alphabet (ISWA). ISWA is continuously updated and is available as an archive with tens of thousands of images (.png), each representing an SW glyph. Glyphs are used to represent configurations of the body parts involved in performing a sign, as well as contact points among such parts and sign dynamics. Figure 15 shows examples of families. In the context of the work reported here, a transcriber named SWift (SignWriting improved fast transcriber) was developed to allow users to compose "written" signs using SignWriting glyphs (Fig. 16). The final goal is to fully integrate it in DELE, to allow both an alternative code for contents and to support personal communication. It will be both a tool provided through an appropriate module to be combined with the others during story design, within chat, forum, etc., as both an authoring tool and a communication tool for learners.

In the metamodel for SW of Fig. 17, an *Utterance* is defined as an instance of a *Concept*, where a same concept can be expressed using many different expressions. Utterances are formed by smaller entities namely *Sign* elements, as represented by the association between these two classes. It is worth noticing that *Sign* elements do not necessarily correspond to words in a verbal language. As a matter of fact, they can, and often do, express an overall articulated thought (sentence) [9]. In other words, a single sign may express a whole short sentence or structured concept. A sign is produced by one or more *Occurrence* of some *Glyph,* representing the specification of a physical manifestation of an individual traceable element. While the term *glyph* is more typical of the definition of SW, it is also used to describe atomic elements common to both SW and SL specifications.

[16] http://www.movementwriting.org/symbolbank/.

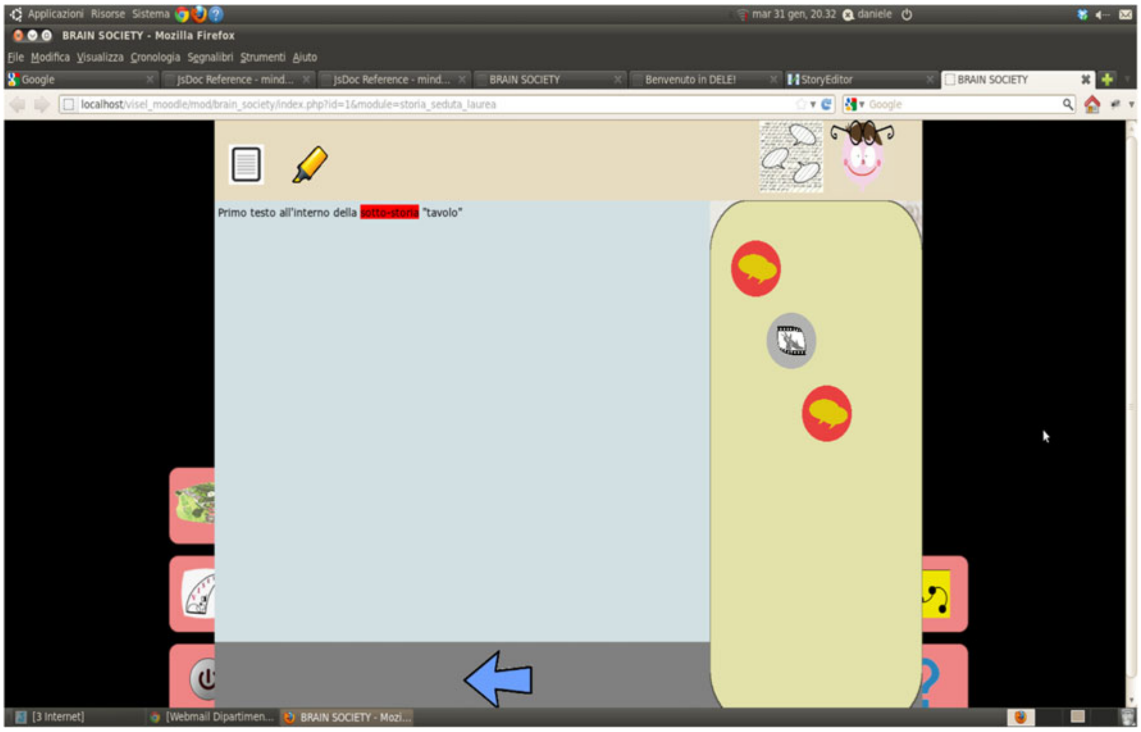

Applicazioni Risorse Sistema
mar 31 gen, 20.32 daniele
BRAIN SOCIETY - Mozilla Firefox
File Modifica Visualizza Cronologia Segnalibri Strumenti Aiuto
Google
JsDoc Reference - mind...
BRAIN SOCIETY
Benvenuto in DELE!
StoryEditor
localhost/visel_moodle/mod/brain_society/index.php?id=1&module=storia_seduta_laurea
Primo testo all'interno della sotto-storia "tavolo"
[3 Internet]
[Webmail Dipartimen...
BRAIN SOCIETY - Mozi...

Fig. 12 Text highlighting indicates the presence of a network of insights

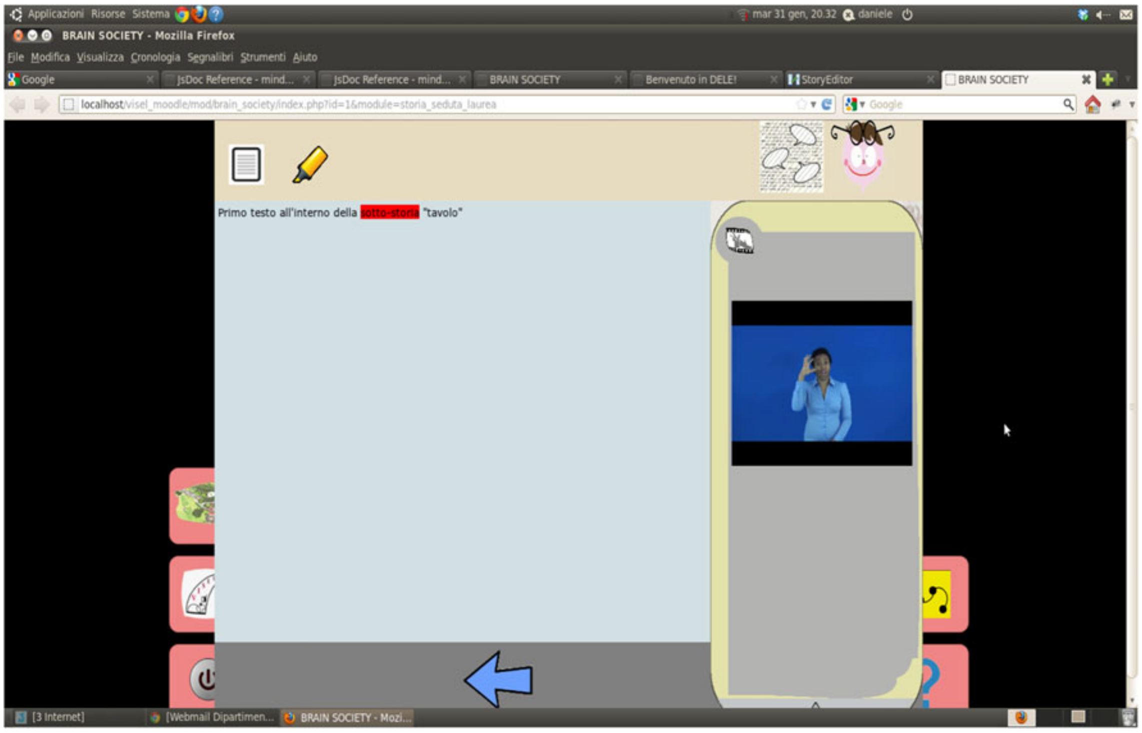
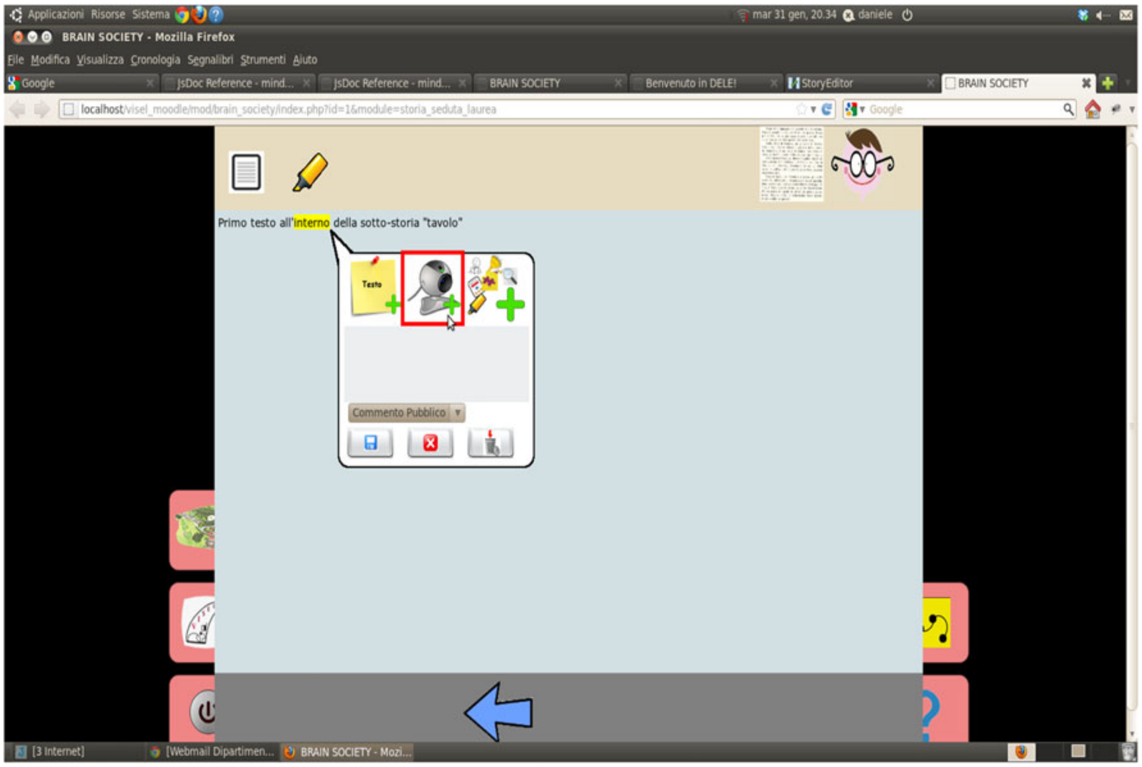

Applicazioni Risorse Sistema
mar 31 gen, 20.32 daniele
BRAIN SOCIETY - Mozilla Firefox
File Modifica Visualizza Cronologia Segnalibri Strumenti Aiuto
Google
JsDoc Reference - mind...
BRAIN SOCIETY
Benvenuto in DELE!
StoryEditor
localhost/visel_moodle/mod/brain_society/index.php?id=1&module=storia_seduta_laurea
Primo testo all'interno della sotto-storia "tavolo"
[3 Internet]
[Webmail Dipartimen...
BRAIN SOCIETY - Mozi...

Fig. 13 One of the insights is shown, in this case a SL video

The metamodel sets an important difference between the specification of a glyph and its occurrences. This is motivated by two facts: first, the ISWA is composed by tens of thousands of glyphs, so there might be glyphs not occurring in any sign, but which are worth being stored and coded, because they might be useful in the future; second, a


Applicazioni Risorse Sistema
mar 31 gen, 20.34 daniele
BRAIN SOCIETY - Mozilla Firefox
File Modifica Visualizza Cronologia Segnalibri Strumenti Aiuto
Google
JsDoc Reference - mind...
JsDoc Reference - mind...
BRAIN SOCIETY
Benvenuto in DELE!
StoryEditor
BRAIN SOCIETY
localhost/visel_moodle/mod/brain_society/index.php?id=1&module=storia_seduta_laurea
Google
Testo
Commento Pubblico
[3 Internet]
[Webmail Dipartimen...
BRAIN SOCIETY - Mozi...

Primo testo all'interno della sotto-storia "tavolo"

Fig. 14 Addition of a personal comment

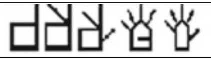
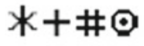
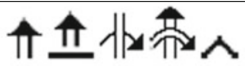
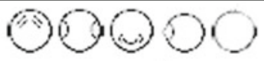
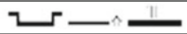
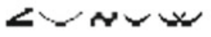
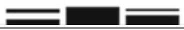
Glyph examples	Family description
	Hand configurations
	Contacts: where, how and how many times a hand comes into contact with a body part
	Movements (of hands, wrists, forearms, etc.)
	Head (expressions, movements, etc.)
	Shoulders, arms, bust
	Dynamics and movement coordination
	Punctuation

Fig. 15 Some examples of families of glyphs

specification also encompasses the admissible variations which can be applied to each individual occurrence. In particular, a *Category* is an aggregate of glyphs subject to some constraint on each possible occurrence and the body parts and movements that can be used to generate their occurrences. Depending on the adopted concrete specification of the lexicon of signs, a category can be organized into sub-categories. Figure 17 lists the collection of categories associated with the ISWA definition, for example, *Configuration, ForearmMovement, HandMovements*.

An example of sub-categorization in the ISWA definition (not shown in the metamodel, but which has been used in SWift) is the specialization of the *HandMovement* category as *StraightHandMovement* and *CircularHandMovement*.

In Sutton's original proposal for a concrete presentation of SW [31], of which the proposed metamodel is a refinement, glyphs were organized according to the following hierarchy. It is presented below to give an idea of the complexity of the dimensions involved.

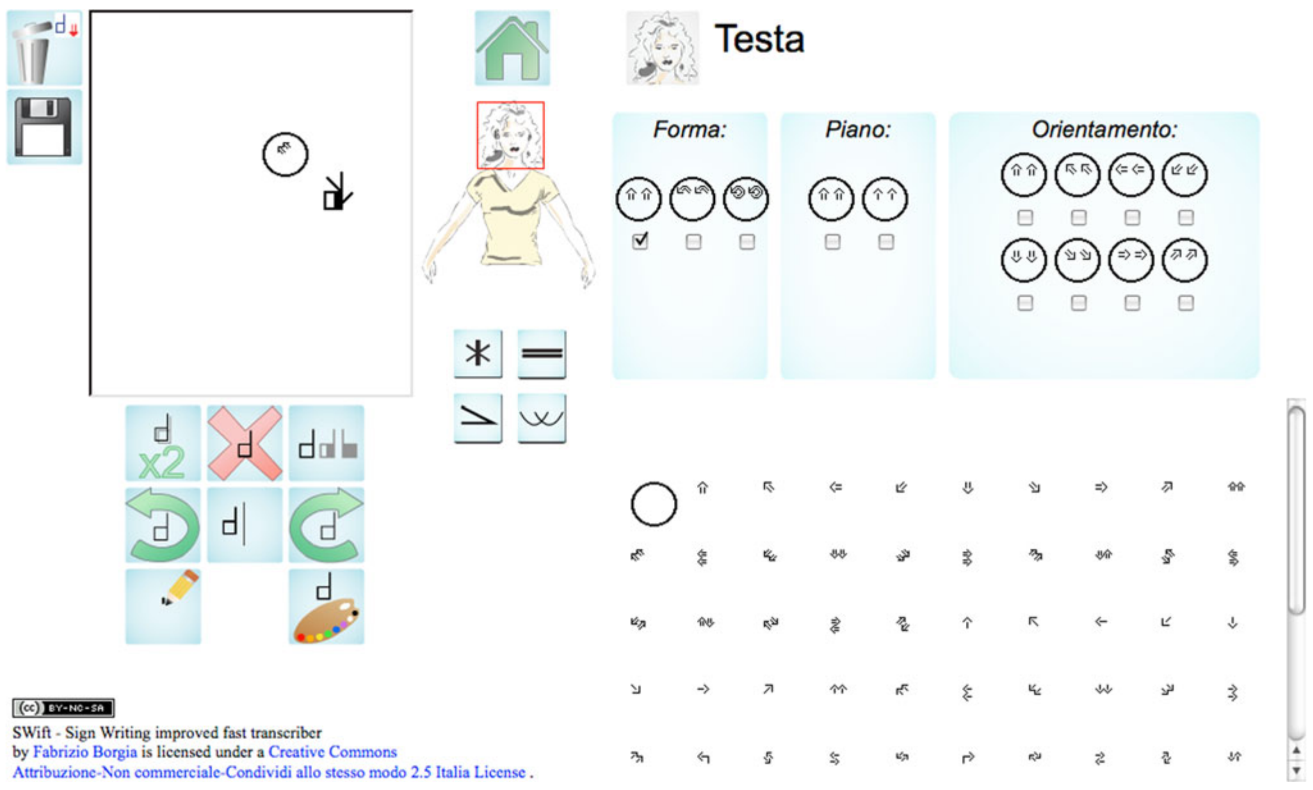
Category: it distinguishes anatomical areas and other elements such as punctuation and contacts: configurations, movements, head and face, body, dynamics and rhythm, punctuation and advanced annotation.

Group: each category is divided into groups at most 10, distinguishing different areas. Groups in a category can be heterogeneous, for example, a single category gathers contacts and all movements (of hands, forearms, wrists and fingers).

Base symbol: identifies a specific glyph in a group.

Variation: distinguishes different manifestations of some symbols; as an example, for the symbol representing the bended forefinger at knuckle, the two possible variations code the difference in angle of the knuckle.

Filling: they identify modifications of the same base symbol; for example, fillings in configurations distinguish the visible side of the hand and the plane where the sign is performed: there are 6 different fillings, corresponding to



SWift - Sign Writing improved fast transcriber
by Fabrizio Borgia is licensed under a Creative Commons
Attribuzione-Non commerciale-Condividi allo stesso modo 2.5 Italia License .

Fig. 16 A screenshot of swift

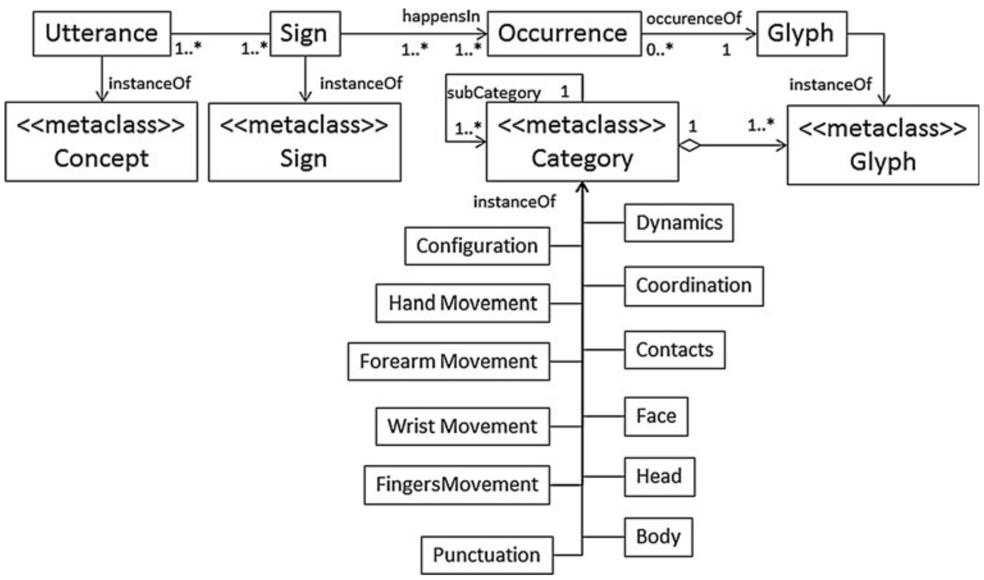


Fig. 17 Metaclasses and classes in a metamodel for Sign Languages and SignWriting

whether the palm, the edge or the back of the hand are seen, and whether the hand rests on the vertical or horizontal plane.

Rotation: as for fillings, they identify modifications of a same base symbol; as an example, in some configurations rotation allows users to distinguish hand orientations, that is, how it is turned and the used hand (left or right).

The main difficulty in defining a concrete syntax for representations of SLs is their four-dimensional quality (one temporal and three spatial dimensions), allowing a high degree of freedom in arranging the base structures that express signs in two dimensions. As a consequence, the SWift interface allows great freedom in the composition and characteristics of aggregated glyphs. The proposal of an abstract syntax through the metamodel drafted in

Fig. 17 is at the basis of Swift and can serve as a guide to build a software framework where signs might be synthesized as well as analyzed.

A first prototype of SWift has been implemented in a contextual design process [2] with a group of deaf users. This method requires that a cross-functional team composed by designers, usability professionals, developers and also customers, interact within the real users' setting during all design phases, in order to reach a common understanding and to agree on users' needs and on how to design a system for them. Regarding SWift design, the resulting interaction pattern reflects the absence of particular limitations on the sign composition. The user can select a specific body part and is introduced to the set of variations and rotations provided for the glyphs in that group. Different aspects are represented by different Choose Boxes. A specific element can be chosen from each of them, so that the set of glyphs to choose is reduced accordingly. Once a glyph has been chosen, it can be dragged and dropped in the board space.

Despite having designed and developed SWift in tight collaboration with deaf researchers, it was decided to also run a first phase of systematic usability testing of SWift. This was considered necessary for a further extensive validation of the application. A usability test was conducted by adapting the "Think Aloud Protocol" to the needs of deaf people. This "adaptation" was in fact a major upheaval in the structure of the test, starting from spatial configuration of the test, the mode of interaction with the participant, the decision to make available any content necessary for the test (typically a list of tasks and the welcome message) in SL, besides spoken language. Deaf users cannot actually "think aloud". They rather communicate through SL and often demonstrate a higher variability in their face expression than other users. Roberts and Fels [26] suggest the setting shown in Fig. 18.

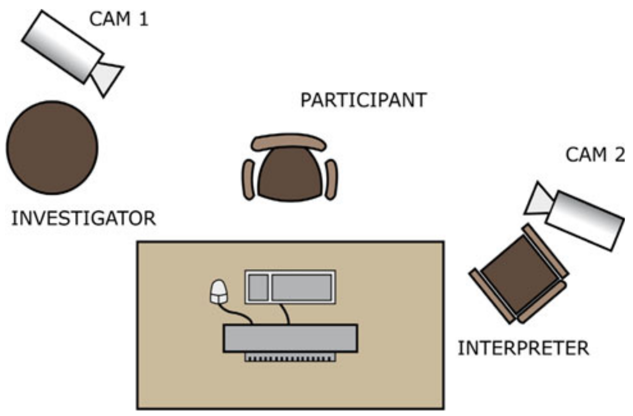


Fig. 18 Experimental setting by Roberts and Fels [26]

CAM1 records a rear view of the participant, the computer screen and the interpreter. CAM2 records the front view of the participant and the investigator. The two recordings must be analyzed and synchronized. Furthermore, when dealing with two separate videos, it is difficult to maintain a synoptic view of what is happening during the test. A system using a single camera was adopted, introducing the use of a projector. The computer screen is projected on the wall, and a single camera records anything that is worth of attention. The test was structured in three phases.

- The welcome time—the participant is greeted and briefed by a screen containing a signed video and its transcription.
- The sign-aloud test—this is the most important part of the procedure: the participant is asked to perform a list of tasks on SWift; during this phase, the participant will be asked to sign anything that comes to her/his mind.
- The final usability questionnaire—adapted from QUIS questionnaire [28].

Of course, the second phase deserved special attention. Users need reminders for which task is being performed, what needs to be done, etc. The task list should be available in both Verbal Language (VL) and SL. It was chosen to involve an interpreter because the possibility of interaction between the participants and the interpreter increases correct understanding of the tasks. In particular, the interpreter always provided a task translation in SL at the beginning of each task. Required tasks included both basic actions, such as inserting a random glyph, or looking for a particular one, and complex ones, such as composing an assigned sign. In the third phase, the participant was asked to answer a usability questionnaire both in SL and in VL. The questionnaire was designed adapting the QUIS usability questionnaire [28] to the specific application and to the needs of deaf users, to stimulate participants to express their opinions. In particular, each (simplified) written question was accompanied by the corresponding SL clip. A preliminary test session was conducted with ten deaf users. The obtained results were very encouraging and will be used to improve some interface aspects. Details are given in the following. It is worth considering that the typical problems of user recruitment for tests are amplified by the peculiarities of the deaf condition, especially since knowledge of SignWriting is still limited, and by the need of a SL interpreter. The possibility of running a modified version of the tests online is also currently being investigated, and it is planned to perform a more extensive evaluation in the next future.

The low number of errors made by participants in the use of interface buttons confirmed the quality of most of the design choices. Some doubts persist on the suitability of the wastebasket icon since it was seldom used. As for the navigation, the lack of any glyph in the home screen was misleading for most participants since they expected to find them. Moreover, the use of graphics different from glyphs as icon for some button or as label for some choice boxes raised some problems since users tended to interpret them in a way different from glyphs or to ignore them. When arriving at a specific set of Choose Boxes, used to identify groups of glyphs related to a specific trait, many deaf users preferred to make only one choice, instead of one for each Box as allowed by the interface. This may be due to the fact that the possibility of choosing one option from each box should be better signaled. It is planned to devise a way to make this possibility clearer. At the end of the test session, most users expressed appreciation for the modalities chosen for the test, in particular for the final questionnaire. The overall results were very satisfying. They underlined precise trends for specific aspects of the application, confirming the reliability of the obtained responses.

8 Conclusions

In this paper, a model-driven approach to developing UIs for deaf people has been presented. A fully iconic page structure is proposed to enhance deaf people's motivation while navigating in virtual environments. In fact, the iconic modality aims at leveraging the deaf-peculiar visual way of grasping information.

This iconic structure is applied to the pages of story-based learning paths, and the StoryEditor visual editor has been presented as a powerful tool for manipulating the two levels of stories description, that is, iconic and diachronic.

Finally, the written representation of Sign Languages has been taken into account, proposing a metamodel for the SignWriting code extendible to Sign Languages in general and which can be the basis for incorporating specifications for this form of interaction into UsiXML.

References

1. Antinoro Pizzuto, E. et al.: Language resources and visual communication in a deaf centered multimodal E-learning environment: issues to be addressed. In: Proceedings of the LREC 2010, pp. 18–23 (2010)
2. Beyer, H., Holtzblatt, K.: Contextual design. Interactions **6**(1), 32–42 (1999)
3. Bolognesi, T., Brinksma, E.: Introduction to the ISO Specification Language LOTOS. Comput. Netw. ISDN Syst. **14**(1), 25–59 (1987)
4. Bottoni, P., et al.: DELE: a deaf-centered E-learning environment. Chiang Mai J. Sci. **38**, 31–57 (2011)
5. Bruner, J.S.: The narrative construction of reality. Critical Inquiry **18**(1), 1–21 (1991)
6. Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., Vanderdonckt, J.: A unifying reference framework for multi-target user interfaces. Interact. Comput. **15**(3), 289–308 (2003)
7. Capuano, D., et al.: A deaf-centred e-learning environment (DELE): challenges and considerations. J. Assist. Technol. **5**(4), 257–263 (2011)
8. Cox, S. et al.: TESSA, a system to aid communication with deaf people. In: Proceedings of the 5th SIGCAPH. pp. 205–212. ACM, New York (2002)
9. Cuxac, C.: French sign language: proposition of a structural explanation by iconicity. In: Braffort, A., Gherbi, R., Gibet, S., Teil, D., Richardson, J. (eds.) Gesture-Based Communication in Human-Computer Interaction. Lecture Notes in Computer Science vol. 1739/1999, pp. 165–184 (1999)
10. Debevc, M., Safaric, R., Golob, M.: Hypervideo application on an experimental control system as an approach to education. Comput. Appl. Eng. Education **16**(1), 31–44 (2008)
11. Efthimiou, E., Fotinea, S.-E., Vogler, C., Hanke, T., Glauert, J., Bowden, R., Braffort, A., Collet, C., Maragos, P., Jérémie Segouat, J.: Sign language recognition, generation, and modelling: a research effort with applications in deaf communication. *Universal Access in Human-Computer Interaction. Addressing Diversity. Lecture Notes in Computer Science*, vol. 5614/2009, pp. 21–30 (2009)
12. Elliott, R., et al.: An overview of the SiGML notation and SiGMLSigning software system. Proc. LREC **2004**, 98–104 (2004)
13. Fajardo, I., Vigo, M., Salmeron, L.: Technology for supporting web information search and learning in sign language. Interact. Comp. **21**(4), 243–256 (2009)
14. Fels, D.I., Richards, J., Hardman, J., Lee, D.G.: Sign language web pages. Am. Ann. Deaf. **151**(4), 423–433 (2006)
15. Fotinea, S.-E., Efthimiou, E., Caridakis, G., Karpouzis, K.: A knowledge-based sign synthesis architecture. UAIS **6**(4), 405–418 (2008)
16. Göhner, P. et al.: Integrated accessibility models of user interfaces for IT and automation systems. In: *Proceedings of CAINE 2008*, pp. 280–285 (2008)
17. Haesen, M. et al.: Using storyboards to integrate models and informal design knowledge. In: Hussmann, H., Meixner, G., Zuehlke, D. (eds.), Model-Driven Development of Advanced User Interfaces. Springer, New York pp. 87–106 (2011)
18. Johnson, M.: The Meaning of the Body. University of Chicago Press, Chicago (2007)
19. Kipp, M., Nguyen, Q., Heloir, A., Matthes, S.: Assessing the deaf user perspective on sign language avatars. In" *Proceedings of the 13th International ACM SIGACCESS Conference on Computers and Accessibility (ASSETS '11)*, pp 107–114. ACM, New York, NY, USA
20. Lakoff, G., Johnson, M.: Metaphor We Live By. University of Chicago Press, Chicago (1980)
21. Limbourg, Q., Vanderdonckt, J.: UsiXML: a user interface description language supporting multiple levels of independence. In: Matera, M., Comai, S. (eds.) Engineering Advanced Web Applications, pp. 325–338. Rinton Press, Paramus (2004)
22. McDrury, J., Alterio, M.: Learning Through Storytelling in Higher Education: Using Reflection and Experience to Improve Learning. Dunmore Press, Palmerston North (2002)
23. Paternò, F.: Model-Based Design and Evaluation of Interactive Applications. Springer, New York (1999)
24. Paternò, F., Santoro, C., Spano, L.D.: MARIA: a universal, declarative, multiple abstraction-level language for service-oriented applications in ubiquitous environments. ACM TOCHI. **16**(4) (2009)
25. Perfetti, C. A., Sandak, R.: Reading optimally builds on spoken language: Implications for deaf readers. J. Deaf Stud. Deaf Education **5**, 32–50 (2000)

26. Roberts, V.L., Fels, D.I.: Methods for inclusion: employing think aloud protocols in software usability studies with individuals who are deaf. Int. J. Human-Comput. Stud. **64**(6), 489–501 (2006)
27. Sharma, R. et al.: Speech-gesture driven multimodal interfaces for crisis management. In: Proceedings of the IEEE, pp. 1327–1354 (2003)
28. Slaughter, L., Norman, K.L., Shneiderman, B.: Assessing users' subjective satisfaction with the information system for youth services (isys). In: VA Tech Proceedings of Third Annual Mid-Atlantic Human Factors Conference, pp. 164–170 (1995)
29. Spano, L.D.: A model-based approach for gesture interfaces. In: Proceedings of EICS 2011, pp. 327–330. ACM, New York (2011)
30. Stanciulescu, A.: A Methodology for Developing Multimodal User Interfaces of Information System. Ph.D. thesis, Université Catholique de Louvain, Louvain-la-Neuve, Belgium (2008)
31. Sutton, V.: A way to analyze American Sign Language and any other Sign Language without translation into any spoken language. In: National Symposium on Sign Language Research and Teaching (1980)
32. Szechter, L.E., Liben, L.S.: Parental guidance in preschoolers' understanding of spatial-graphic representations. Child Dev. **75**(3), 869–885 (2004)
33. Van Hees, K., Engelen, J.: Non-visual access to GUIs: leveraging abstract user interfaces. In: Proceedings of ICCHP'2006, LNCS 4061, pp.1063–1070. Springer, New York (2006)