5 Un nuovo quadro teorico di progettazione e analisi

In questo capitolo si delineerà un quadro teorico di riferimento da applicare alle singole funzionalità del case study *JaLea*, che permetta di analizzare a vari livelli di astrazione quali strategie sono state utilizzate per motivare gli utenti all'utilizzo dell'applicativo in autonomia. Questa metodologia si basa su due assi: un asse orizzontale diacronico, e un asse verticale sincronico multilivello.

5.1 Asse diacronico e asse sincronico a livelli multipli di astrazione

Il primo asse, quello diacronico, è basato sul modello di sviluppo ADDIE ricavato dalla teoria dell'instructional design. Il secondo asse, sincronico, prevede differenti livelli di astrazione ed è basato sul framework di Garrett (2007).

DOI 10.30687/978-88-6969-554-4/005

Il quadro di riferimento descritto nel presente capitolo risulta essere molto utile nell'analisi di progetti particolarmente vasti e complessi come *JaLea*, in quanto offre una metodologia per analizzarne le singole funzionalità in ottica di experience design. In questo capitolo pertanto, dopo aver descritto il quadro di riferimento, questo verrà applicato, a scopo di esempio, a due sole funzionalità selezionate di *JaLea*.

5.1.1 Asse diacronico - ADDIE

Come indicato nel capitolo 3, ADDIE è un acronimo per: analysis (analisi), design (progettazione), development (sviluppo), implementation (implementazione) e evaluation (valutazione). Questo modello, utilizzato normalmente in ambito di progettazione di corsi di apprendimento e formazione, può essere adottato in ambito di Instructional System Design (ISD) per il ciclo di progettazione dell'applicativo attraverso i seguenti step.

Analysis vengono identificate le caratteristiche degli utenti principali che utilizzeranno il sistema, e/o gli obiettivi del progetto o della funzionalità da sviluppare.

Design viene pianificato lo sviluppo delle funzionalità definite nel processo di analisi, creando una roadmap e una serie di milestone su cui suddividere il progetto.

Development in base alle valutazioni sulla tipologia di utenti che utilizzerà il prodotto e al tipo di contenuti che si desidera veicolare, viene creato un progetto e si definiscono i modelli di navigazione. In questa fase, inoltre, vengono anche decise le tecnologie da utilizzare per lo sviluppo dell'applicativo.

Implementation viene sviluppato l'applicativo in base alla pianificazione effettuata nelle fasi precedenti, e viene sottoposto a prove e controlli che ne verifichino la stabilità.

Evaluation in base all'utilizzo da parte degli studenti e dei collaboratori, vengono raccolti dei feedback che permettono di capire come procedere con la pianificazione e lo sviluppo ulteriore dell'applicativo. Pertanto, dopo questa fase il ciclo riparte dalla fase 'Analysis'.

La raccolta dei feedback avviene in tempi e modalità differenti. Se dopo la fase di implementazione si è soliti chiedere agli studenti, tramite questionari, le opinioni relative all'esperienza d'uso dell'applicativo, durante le fasi di sviluppo è buona norma discutere l'andamento dello sviluppo analizzando quanto prodotto fino a quel momento tra i membri del team, attraverso riunioni concordate.

Nel caso di *JaLea*, ad esempio, l'attività di sviluppo è iniziata con la creazione dell'area di backend in quanto era necessario permettere al content manager di inserire i materiali. I primi feedback pertanto, provengono proprio dai collaboratori che si occupano di inse-

rire i materiali, che confermano o meno la validità di un determinata interfaccia o logica di business (§ 5.3.7).

Dopo aver definito a grandi linee la struttura di tutto il progetto, le fasi di sviluppo, implementazione e valutazione si procede per milestone. Il progetto, cioè, viene diviso in più fasi, che comprendono la progettazione e lo sviluppo di alcune funzionalità, ognuna di queste viene verificata dai membri del team, dai collaboratori e da studenti volontari, che forniscono feedback immediati per capire immediatamente se ci sono problemi d'uso gravi. Solo una volta che questo processo è assestato, si può ottenere un feedback più sostanziale da parte degli utenti generici, anche attraverso sondaggi qualitativi.

5.1.2 Asse a livelli di astrazione - i 5 livelli di Garrett

Mentre il livello ADDIE è utile per definire un *metodo* di sviluppo pratico di tipo diacronico, è meno utile per identificare le varie fasi di sviluppo del progetto in ottica di experience design. Come è stato indicato nel capitolo 4, un'applicazione Web è la realizzazione digitale di un prodotto complesso che richiede l'ideazione di strategie di usabilità in ambiti differenti.

Conviene pertanto utilizzare anche un framework metodologico che permetta di rappresentare a diversi livelli di astrazione l'applicativo o le sue funzionalità. Si può ritenere adatto a questo scopo il framework proposto da Garrett (2011), che identifica 5 livelli: Strategia (*Strategy*), Dominio (*Scope*), Struttura (*Structure*), Scheletro (*Skeleton*), Superficie (*Surface*).

Ogni livello fornisce un modello concettuale attraverso il quale è possibile analizzare come implementare una o più strategia di esperienza utente per risolvere problemi di usabilità, motivazione e manutenibilità.

Ogni piano è dipendente dal piano sottostante, quindi la superficie dipende dallo scheletro, che a sua volta dipende dalla struttura, che dipende dal dominio, il quale a sua volta dipende dalla strategia. Quando le decisioni prese non si allineano con i piani superiori e sottostanti, si possono verificare discrepanze di pianificazione che portano inevitabilmente ad allungamenti nei tempi di sviluppo, con conseguente aumento dei costi e mancato raggiungimento degli obiettivi preposti.

Al fine di una analisi più dettagliata dei processi che avvengono a ciascun livello, Garrett (2011) identifica due aree di interesse all'interno di ognuno di questi, una dal punto di vista dell'information design, e l'altra dal punto di vista dell'information architecture.

Questi due punti di vista richiamano in un certo senso il modo con cui è concepito il Web 1.0 e 2.0, la cui differenza è stata ampiamente descritta nel capitolo precedente.

Tim Berners-Lee infatti, inventò il World Wide Web per avere un mezzo per informare la comunità di fisici di cui faceva parte di nuove scoperte e informazioni (Berners-Lee 1992). Il Web 1.0 pertanto, permetteva unicamente di fornire informazioni. Grazie però alla popolarità che ottenne nel corso degli anni e allo conseguente sviluppo di tecnologie correlate, il Web diventò in grado, non solo di veicolare informazioni, ma anche di raccoglierle e manipolarle e di adattare le informazioni da presentare in base alle richieste degli utenti (Web 2.0).

Nonostante la trasformazione del concetto di pagina Web prettamente informativa (Web 1.0) in applicazione (Web 2.0), l'utilizzo del Web come sistema prettamente informativo continuò a fiorire nel formato di testate giornalistiche, riviste online e simili.

Esistono quindi due visioni differenti: la prima visione considera il Web come una piattaforma che veicola informazioni, e relega la gestione del materiale multimediale e dei contenuti al mondo dell'editoria e dei media. Questo concetto si concretizza nella figura del content manager, che presiede alle attività di gestione dei contenuti attraverso l'area di backend.

L'altra visione è legata all'ingegneria del software, tendente quindi a l'implementazione di tecnologie, o di evoluzioni strutturali dell'applicativo.

È necessario, quindi, considerare ogni livello del framework da due prospettive differenti: una che analizza il prodotto dal punto di vista delle funzionalità (aspetto ingegneristico) e una che lo analizza dal punto di vista delle informazioni (aspetto contenutistico). La differenza del punto di vista dell'analisi di ogni livello, porta all'utilizzo di competenze differenti, come esemplificato dallo schema di Garrett della figura **5.1**.

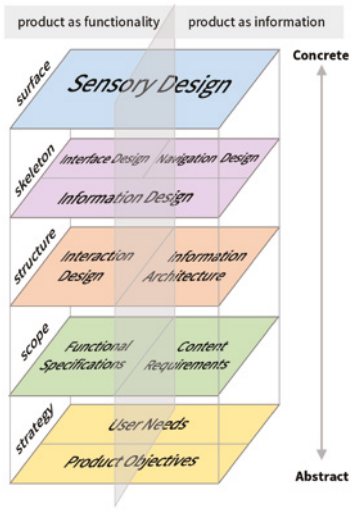


Figura 5.1 Framework di Garrett (2011)

Strategy (Strategia)

Il piano della strategia riguarda i fini con i quali chi crea e gestisce l'applicativo si pone (siano essi di tipo economico, divulgativo o educativo) e la percezione dei fini cui mirano i futuri utilizzatori. Questo piano è assimilabile alla fase di 'analisi' del modello ADDIE. In questo piano cioè, vengono analizzate le finalità del prodotto (*product objectives*) e le necessità degli utenti (*user needs*).

I seguenti quattro piani sono assimilabili alle fasi di pianificazione e sviluppo del modello ADDIE, ma permettono di analizzare a differenti livelli le strategie di sviluppo.

Scope (Dominio)

Vengono definite le specifiche funzionali (*functional specifications*) e i requisiti di contenuto (*content requirements*). Attraverso le specifiche funzionali viene descritto come ogni funzionalità deve essere presentata a livello formale. Ad esempio: «la sezione video di *JaLea* deve presentare un'area superiore e un'area laterale, o inferiore, a seconda della grandezza dello schermo del dispositivo da cui vi si accede», «La sezione laterale/inferiore è dedicata alla trascrizione in giapponese del testo, ove devono essere presenti le trascrizioni sia in *hiragana* che *rōmaji*».

Attraverso la definizione dei requisiti di contenuto, invece, si definiscono i requisiti tecnici che i contenuti devono possedere, ad esempio come vengono inseriti i video nella piattaforma (caricati dal server interno o da YouTube), o come sono pianificate le attività per inserire e gestire i contenuti dal punto di vista tecnico.

Structure (Struttura)

A questo livello viene definita la struttura del prodotto. Dal punto di vista delle informazioni, si parla di information architecture, mentre dal punto di vista delle funzionalità, di *interaction design*.

Attraverso l'information architecture, vengono descritte in modo sequenziale le possibili attività dell'utente sull'interfaccia, le relative logiche di business che regolano le azioni sull'interfaccia, la mappatura del salvataggio dei dati nelle tabelle del database.

Attraverso l'interaction design viene descritto come il sistema risponde agli stimoli dell'utente. Come già trattato nel capitolo 4 relativamente al concetto di embodiment, ogni volta che un programma viene utilizzato, si instaura tra questo e il suo utilizzatore una relazione: l'utente invia un segnale al sistema attraverso un dispositivo hardware, e il sistema risponde in un determinato modo.

In questa relazione due elementi sono fondamentali: 1) il modello mentale su cui si basa l'interfaccia e 2) il feedback (avvisi e gestione degli errori).

Il modello mentale è la scorciatoia cognitiva usata dall'utente per cercare di comprendere il funzionamento dell'interfaccia appena entra in relazione con questa. Questo modello si differenzia dal modello implementativo che descrive l'interfaccia nella sua totalità, compresi gli elementi non percepibili dall'utente (struttura del codice e la logica di business), e dal modello rappresentativo, con cui gli sviluppatori presentano effettivamente l'interfaccia all'utente.

Se l'interfaccia è realizzata con un modello rappresentazionale vicino al modello mentale degli utilizzatori, l'utente sarà in grado di relazionarsi subito con questa, senza ulteriore sforzo cognitivo.

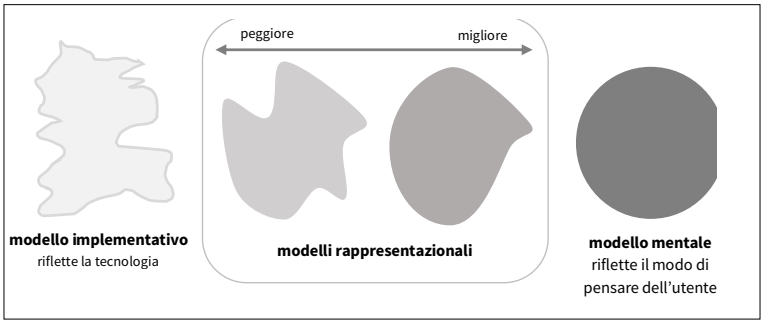


Figura 5.2 Immagine dei modelli cognitivi principali identificati nel processo di embodiment (Garrett 2011)

Relativamente al concetto di feedback, questo è stato trattato nel capitolo 3 in ambito di usabilità degli artefatti (Norman 2013). Il feedback è per Norman un qualsiasi segnale che permetta all'utilizzatore di capire che l'oggetto funziona secondo le sue richieste. Questo concetto, applicato allo sviluppo di una applicazione Web, indica un qualsiasi segnale all'interno dell'interfaccia, sia esso visivo o sonoro, che permetta all'utente di capire quale area o funzionalità del programma sia in uso. Un ulteriore uso del feedback è rappresentato dal controllo dell'errore. È fondamentale infatti definire strategie di segnalazione di problemi relativi all'utilizzo errato dell'interfaccia da parte dell'utente, in modo che il flusso d'esperienza di interazione non venga interrotto da un eccessivo sforzo cognitivo di quest'ultimo.

Il controllo degli errori si basa principalmente su due fattori:

1. prevenzione: impedire operazioni errate nell'interfaccia;
2. segnalazione e correzione: segnalazione degli errori ed eventuale suggerimento di correzione

Skeleton (Scheletro)

In questo livello vengono definiti tutti gli elementi dell'interfacce del sistema (*interface design*) quali: i tasti, i campi di inserimento del testo, le tendine di selezione e i selettori *radio* e *checkbox*.

Le principali regole da rispettare nel design delle interfacce, secondo Garrett (2011) sono: a) inserire dove è necessario hyperlink per collegare elementi di pagine differenti; b) permettere attraverso signifier appositi di fornire feedback relativi all'area dell'applicativo in cui l'utente si trova, indicando se possibile l'eventuale struttura gerarchica dell'albero di navigazione; c) visualizzare chiaramente in ogni pagina l'interfaccia di navigazione dell'applicativo.

Surface (superficie)

In questo livello viene organizzata la gestione di tutti gli elementi grafici dal punto di vista dell'uniformità di forma e di eleganza posizionale, nonché la tipografia dei caratteri e la palette di colori.

5.2 Modello semplificato ADDIE-Garrett

Il framework di Garrett è complesso, in quanto creare un'esperienza utente completa di un sistema e-learning seguendolo richiede l'uso di molte discipline: informatica (gestioni sistemi e programmazione), glottodidattica, content designing, interface designing, graphic designing.

Tuttavia, come indicato nel capitolo precedente, un approccio alla creazione sostenibile di un applicativo è possibile analizzando prima di tutto quali risorse, temi grafici, librerie di sviluppo e di organizzazione spaziali dei contenuti sono già disponibili in Internet per l'attività di sviluppo. Ad esempio, è possibile utilizzare un tema grafico che contiene già le funzionalità per rendere l'applicazione responsiva, applicando così competenze avanzate di graphic design senza necessariamente consultare un esperto del settore.

Inoltre, nel caso si consideri il framework di Garrett non ai fini della progettazione ex novo, ma per descrivere le strategie utilizzate per lo sviluppo di determinate funzionalità di un applicativo in ottica di experience design, è possibile utilizzare solo alcuni elementi del framework (soprattutto quelli relativi all'interfaccia) per la creazione di un nuovo modello semplificato, restringendo il campo agli scopi comunicativi necessari. Nel caso della presente monografia, ad esempio, può essere non particolarmente rilevante descrivere la fase di information architecture, che affronta il tema del salvataggio dei dati all'interno di un database. Al contrario, se le logiche di information architecture riguardano l'elaborazione e la presentazione del testo giapponese, sarà necessario descrivere i relativi passaggi tecnici.

Pertanto, il framework di Garrett può essere utilizzato come un modello ideale e adattabile alle varie esigenze di progettazione e descrizione. È possibile ad esempio sostituire il nome di un livello con una descrizione più adatta all'ambiente da descrivere: nel caso di *JaLea* il nome del livello *sensory design* è stato sostituito con il termine 'Tema', in quanto gran parte della grafica (colori, font, posizione degli elementi) viene definita da un tema grafico acquistato preventivamente.

I livelli, così definiti, vengono poi incrociati con l'asse diacronico del processo ADDIE, in modo da ottenere una doppia prospettiva sull'attività di sviluppo temporale e verticale per l'identificazione e la descrizione di strategie di user experience.

Nei prossimi paragrafi, il modello composito e semplificato ADDIE-Garrett viene utilizzato per descrivere funzionalità di *JaLea*.

5.3 Applicazione del modello semplificato ADDIE-Garrett a *JaLea*: agevolare l'inserimento dei contenuti e la creazione di hyperlink

Sulla linea diacronica del modello ADDIE, vengono sovrapposti i livelli ricavati dal framework di Garrett più confacenti ad analizzare la funzionalità descritta di seguito. Vengono tralasciate le fasi relative ai requisiti di contenuto e all'information architecture, in quanto particolarmente tecniche e non interessanti in questo particolare caso.

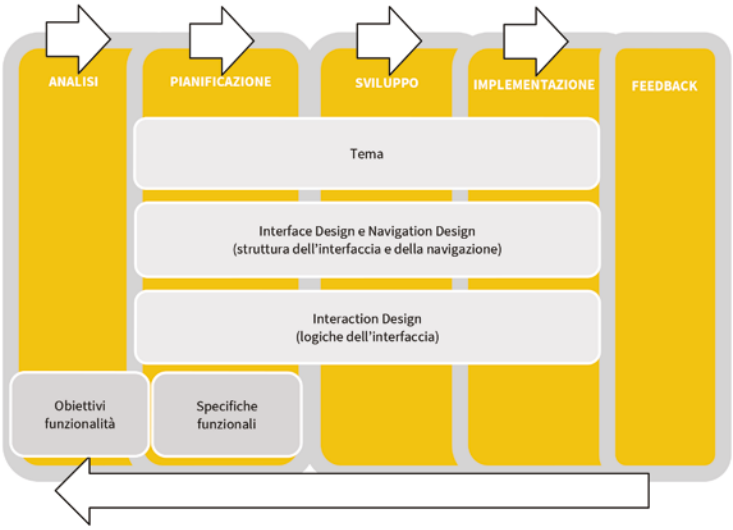


Figura 5.3 Esempio di schema semplificato ADDIE-Garrett

L'ultimo elemento del framework di Garrett, quello relativo al sensory design, è stato sostituito con il termine 'tema' per indicare che l'implementazione di *sensory (graphic) design* è delegata all'utilizzo di un tema grafico preimpostato.

Dopo aver definito il modello, si procede alla descrizione di ogni elemento.

5.3.1 Analisi

L'obiettivo della funzionalità analizzata nel presente paragrafo è la seguente: identificare e implementare un metodo per permettere al docente di inserire e modificare materiale testuale in *JaLea*, secondo una metodologia di lavoro rispettosa del suo tempo e del suo sforzo.

5.3.2 Specifiche funzionali

Come si è scritto, *JaLea* prevede una divisione a due livelli: la 'Descrizione' di ciascuna voce grammaticale, e il 'Dettaglio' dei relativi impieghi. Si prenda, ad esempio に *ni*, particella di caso e coniugazione avverbiale dell'ausiliare pseudo-verbale '*da*/*desu*' **[fig. 5.4]**.

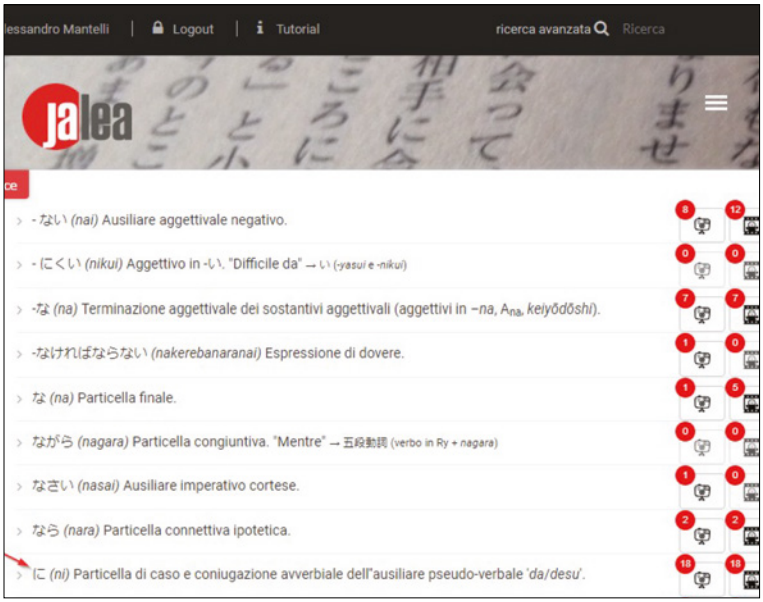


Figura 5.4 *JaLea*. Indice con evidenziata la voce relativa alla particella に *ni*

Cliccando questa voce apparirà la pagina di descrizione, con una spiegazione concisa e immediata della particella.

Sulla sinistra della pagina, sono presenti tre blocchi di testo che indicano 'Significato', 'Descrizione grammaticale', 'Traduzione approssimativa' di tutti i casi d'uso possibile. Sulla destra è invece posizionato un menu verticale che presenta tutti i casi d'uso della particella, chiamato menu di dettaglio. Quest'ultimo è diviso in più blocchi, distinti per tipologia di unità sintattica identificata, o per tipo di ele-

menti elencati. Nel caso della particella に *ni* ad esempio, l'alberatura del menu di dettaglio è la seguente:

Complementi 'complemento d'agente', 'complemento di stato in luogo', 'complemento di moto a luogo', 'complemento di termine', 'complemento di tempo determinato', 'complemento di modo'.

Costruzione 'mutamento'.

Subordinate 'subordinata finale + verbi di moto'

Approfondimenti 'riferimenti temporali', 'differenze con *to*, *kara* e *de*', 'avvertenze per le proposizioni finali con *ni*'.

Elementi correlati Lista degli elementi correlati.

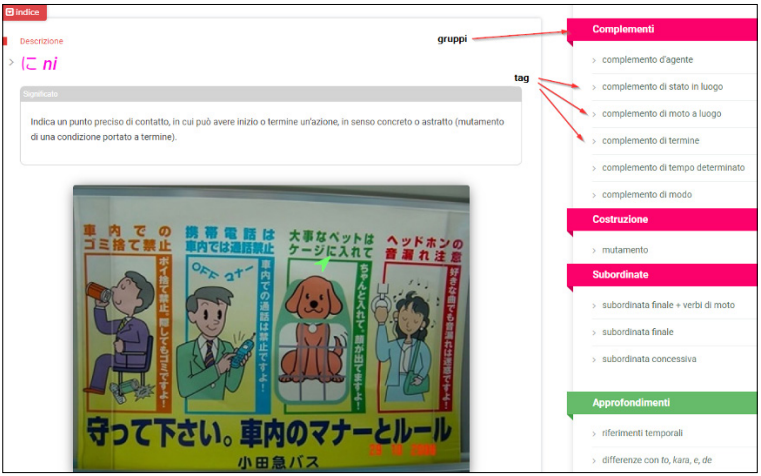


Figura 5.5 Pagina di descrizione della voce に *ni*. Da notare le voci del menu di destra

Cliccando su una delle voci all'interno del menu nel lato destro della pagina di descrizione, appare la relativa pagina di dettaglio. Ad esempio, cliccando 'Complemento di moto a luogo', appare una pagina con esempi e approfondimenti relativo all'utilizzo della particella に *ni* come marcatore del complemento di moto a luogo.

La modalità di presentazione dei contenuti nelle pagine di dettaglio di *JaLea* può variare, ma tendenzialmente è formata da a) una tabella esemplificativa della disposizione dell'elemento grammaticale preso in esame all'interno di una frase; b) frasi di esempio con relativa traduzione, e c) note di approfondimento.

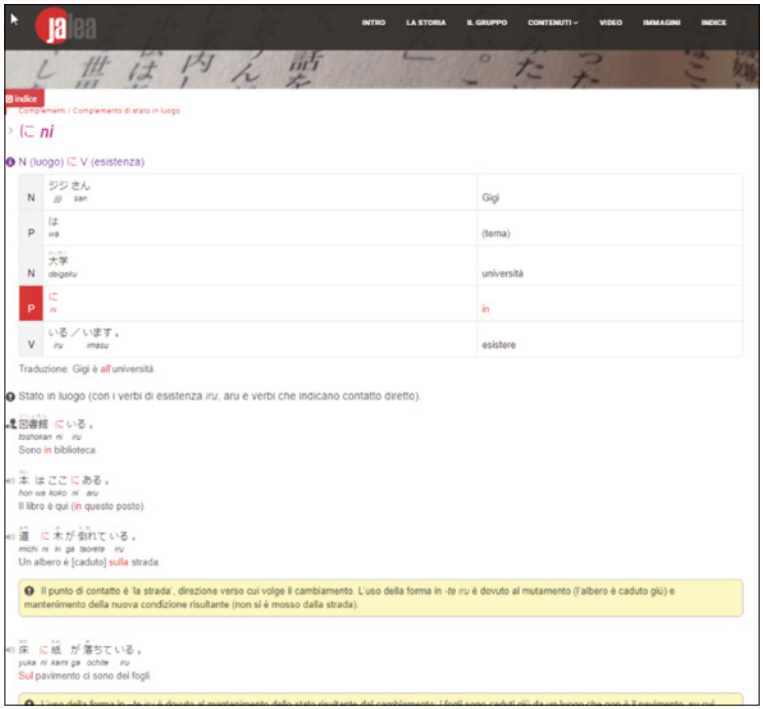


Figura 5.6 Dettaglio della pagina della voce に *ni* come complemento di moto a luogo

Questa metodologia di navigazione, secondo uno schema induttivo (dal generale al particolare), consente al discente di avere una panoramica di base, per poi iniziare il proprio percorso di 'scoperta' degli elementi grammaticali attraverso i link ai vari dettagli.

Le specifiche funzionali descritte di seguito intendono evidenziare quali strategie sono state usate per permettere l'inserimento dei materiali testuali, che descrivono i casi d'uso nella lingua italiana degli elementi grammaticali giapponesi, e la gestione del processo di inserimento dei materiali da parte del content manager.

La classificazione degli elementi grammaticali giapponesi in corrispondenti categorie e possibili funzioni nella lingua italiana, è un'attività più complessa rispetto all'inserimento e alla modifica di testo per spiegazioni ed esempi. Questo perché, mentre la prima attività richiede competenze di classificazione grammaticale e sintattica degli elementi da inserire, la compilazione dei contenuti può essere eseguita facilmente anche da uno studente sotto la supervisione di un docente. Pertanto, in *JaLea* l'attività di classificazione del materiale, divisa in due blocchi chiamati 'gruppi' e 'tag', è eseguibile solo da parte di content manager con credenziali di amministratore.

La funzionalità 'gruppi' permette di definire un comune denominatore sotto il quale raggruppare più tag attribuiti di volta in volta all'elemento da inserire. Questi gruppi possono identificare Complementi, Subordinate o Costruzioni, oppure categorie utili quali 'Avvertenze', 'Utilizzo in vari contesti', 'Approfondimenti'.

I tag, in *JaLea*, vengono utilizzati per marcare le pagine di dettaglio. A seconda dell'elemento grammaticale trattato, i tag possono essere chiamati: 'passivo', 'potenziale', 'comparativo di uguaglianza', ma esistono anche tag chiamati 'differenze tra *to*, *ba*, *tara* e *nara*', 'eccezioni verbi terminanti in *-iru* e *-eru*' relativi alle schede di approfondimento.

Questo tipo di associazione 'uno-a-molti' tra gruppo e tag si traduce nell'area di frontend delle categorie grammaticali e dei relativi elementi di approfondimento.

Una volta definiti gruppi e tag si procederà a creare la scheda 'Dettagli' che conterrà il contenuto da visualizzare. Si presenterà nel sotto-paragrafo seguente come funziona la creazione di una scheda di dettaglio; si descrivono invece ora quali strategie sono state identificate per permettere al content manager di inserire contenuti utilizzando un metalinguaggio semplificato, derivante dal codice HTML creato ad hoc per *JaLea*.

L'HTML è un linguaggio di marcatura[1] con lo scopo principale di contrassegnare testi, link e contenuti con delle etichette (tag) in modo da indicarne la posizione, lo stile e il ruolo all'interno della pagina. Questi marcatori sono soprattutto semantici, ovvero indicano il ruolo o l'organizzazione che si desidera assegnare ai contenuti. I nomi dei tag sono richiusi tra parentesi angolari (<>).[2] Ad esempio:

```
1. <h1>header1</h1>
2. <p>questo è un paragrafo</p>
3. <a href='http://www.google.com'>questo è un link</a>
```

sono tre esempi di tag HTML che marcano un testo come 1) titolo; 2) paragrafo, e 3) link. Si noti per ognuno di questi tag `<h1>`, `<p>`, `<a>` la presenza del relativo tag di chiusura `</h1>`, `</p>`, `</a>`.

Sebbene l'utilizzo dell'HTML non sia particolarmente complesso, apprendere tutti i tag necessari per la creazione di una pagina può richiedere abbastanza tempo. Inoltre, l'utilizzo di troppi tag nel testo che si inserisce, rende quest'ultimo difficile da leggere e da modificare.

1 Sulla questione se considerare l'HTML un linguaggio di programmazione o meno veda ad esempio (Ray 2015)

2 Per informazioni sui tag HTML, «Elementi e tag in HTML»: `https://www.html.it/pag/16030/elementi-e-tag-in-html`.

È stata identificata quindi una strategia per permettere al content manager di inserire il testo utilizzando quasi esclusivamente tag creati ad hoc per funzioni specifiche del sistema *JaLea*, attraverso un'area di testo speciale all'interno dell'interfaccia di backend, predisposta alla creazione delle schede.

In *JaLea* questi tag permettono di definire lo stile di titoli, tabelle, esempi in giapponese con trascrizioni automatiche in *furigana* e *rōmaji*, aree di approfondimento e link automatici alle varie sezioni dell'applicativo. L'area di frontend interpreterà i tag inseriti nel backend e creerà automaticamente la funzionalità indicata dal marcatore.

Di seguito il dettaglio delle specifiche funzionalità che agiscono sull'area di backend e/o frontend.

Backend

1. creazione di funzionalità per creare gruppi e tag con credenziali di amministratore;
2. creazione di funzionalità per l'inserimento di dati visualizzabili nell'area di frontend quali: indice, aree di descrizione e aree di dettaglio;
3. creazione di funzionalità per l'inserimento di tag specifici per il sistema *JaLea* in un'area di testo dedicata.

Frontend

1. creazione delle funzionalità dell'indice e logiche di navigazione. Cliccando una voce dell'indice si passa alla relativa pagina di descrizione;
2. creazione della pagina di descrizione e logiche di navigazione. Cliccando una voce del menu a destra si passa alla relativa pagina di dettaglio;
3. creazione della pagina di dettaglio e logiche di navigazione.

5.3.3 Interaction Design

Le interfacce del backend di *JaLea* sono state create sulla base di un modello CRUD (*Create Read Update Delete*)[3] tradizionale. La schermata principale di ogni menu visualizza una tabella ordinabile alfabeticamente per campo, ogni riga della quale è rappresentata da un record di dati caricati dal database. Quando viene cliccato un record, un form che permette l'inserimento e la modifica dei dati viene visualizzato al posto della tabella. Per inserire un nuovo record si preme il tasto [Add]. Inoltre, attraverso il campo [Cerca] è possibile filtrare la tabella per parola chiave.

3 Definisce un processo attraverso il quale tramite specifiche funzioni è possibile creare, leggere, aggiornare ed eliminare dati. Cf. Code Academy, «What is Crud?»: `https://www.codecademy.com/articles/what-is-crud`.

La tabella è responsiva, ovvero si adatta automaticamente alla dimensione della finestra. Ogni operazione di visualizzazione, inserimento e modifica avviene attraverso AJAX (§ 4.3.2), in questo modo tutte le interazioni con la pagina avvengono in modo fluido (ovvero senza un'alterazione momentanea dell'aspetto grafico) e veloce rispettando il concetto di non interruzione del flusso in ambito di embodiment tra essere umano e artefatto digitale (Triberti, Brivio 2016; Winograd, Flores 2008).

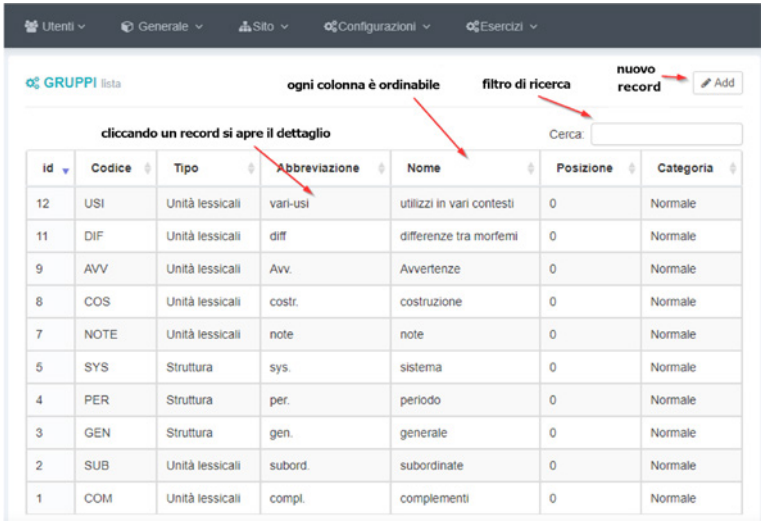


Figura 5.7 *JaLea*, area backend. Lista della pagina Gruppi tag del backend. Colonne ordinabili e funzionalità di ricerca per parola chiave

La struttura della scheda di inserimento e modifica è quella di un form, con una serie di campi responsivi. Alla pressione del tasto [salva], ad esempio, vengono verificati gli eventuali campi obbligatori (evidenziati da un asterisco prima del nome), ed è segnalata in modo chiaro la necessità della loro compilazione. Nel caso non siano presenti errori il sistema procede al salvataggio della scheda.

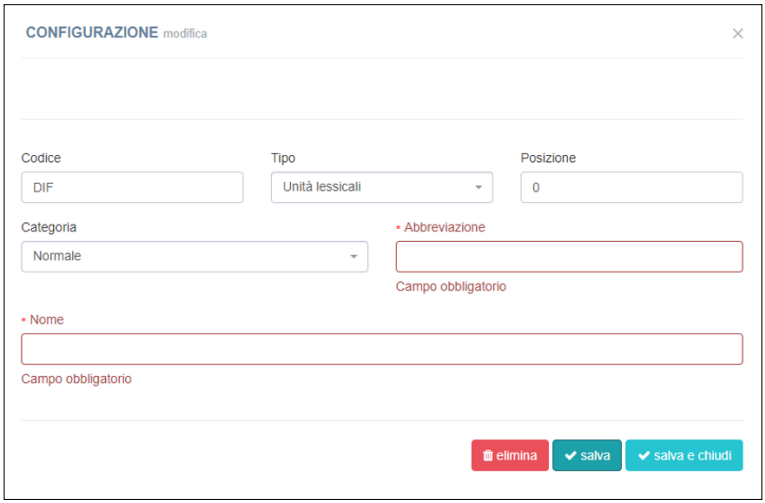
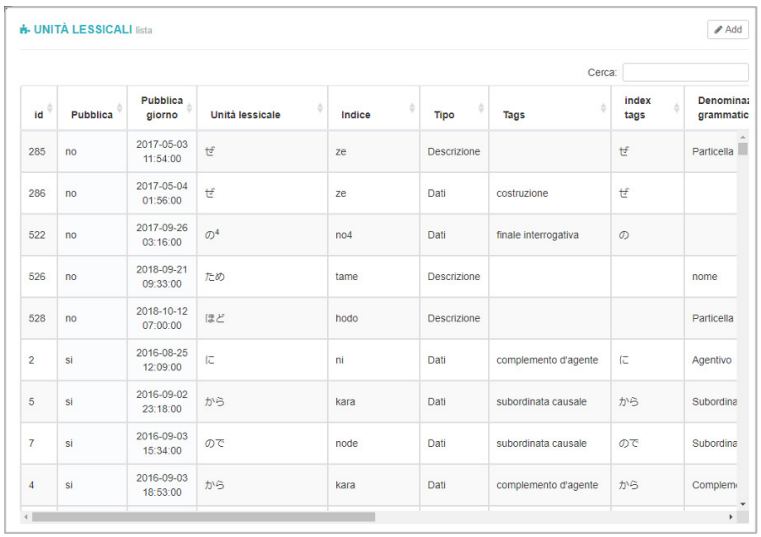


Figura 5.8 *JaLea*, area backend. Scheda di inserimento e segnalazione d'errore

Anche tutte le operazioni di accesso al database avvengono attraverso AJAX e non richiedono il ricaricamento della pagina, pertanto la percezione d'uso è fluida e la velocità del processo di inserimento/modifica delle pagine è ottimizzata al massimo.

Si descrive di seguito il funzionamento dell'interfaccia per la gestione dei contenuti legati alla compilazione delle schede di 'Dettaglio'.



id	Pubblica	Pubblica giorno	Unità lessicale	Indice	Tipo	Tags	index tags	Denomina: grammatic
285	no	2017-05-03 11:54:00	ぜ	ze	Descrizione		ぜ	Particella
286	no	2017-05-04 01:56:00	ぜ	ze	Dati	costruzione	ぜ	
522	no	2017-09-26 03:16:00	の[4]	no4	Dati	finale interrogativa	の	
526	no	2018-09-21 09:33:00	ため	tame	Descrizione			nome
528	no	2018-10-12 07:00:00	ほど	hodo	Descrizione			Particella
2	si	2016-08-25 12:09:00	に	ni	Dati	complemento d'agente	に	Agentivo
5	si	2016-09-02 23:18:00	から	kara	Dati	subordinata causale	から	Subordina
7	si	2016-09-03 15:34:00	ので	node	Dati	subordinata causale	ので	Subordina
4	si	2016-09-03 18:53:00	から	kara	Dati	complemento d'agente	から	Complem

Figura 5.9 *JaLea*, area backend. Elenco delle unita lessicali

La creazione di una nuova scheda di dettaglio avviene a partire dall'area di riepilogo (elenco) delle unità lessicali.

Una volta cliccato il tasto [Add], apparirà una scheda nuova, divisa in cinque tab differenti: 'Principale', 'Attributi', 'Correlati', 'Dettaglio', 'Pointer'. Attraverso le funzioni del primo tab 'Principale', sarà possibile configurare la scheda in modo che le informazioni relative vengano interpretate dal frontend di *JaLea* come un'area di dettaglio. Questo avviene selezionando dal menu a tendina [Tipo], l'opzione 'Dati', e attraverso il menu a tendina [unità lessicali], l'unità lessicale a cui associare la scheda di dettaglio.

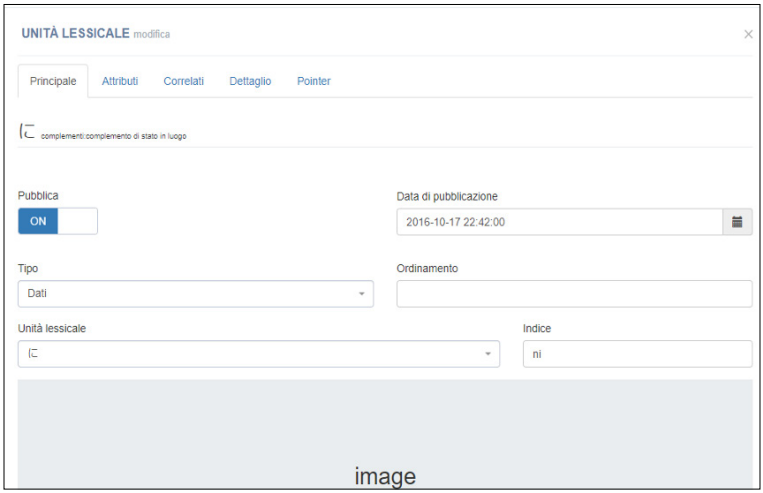


Figura 5.10 *JaLea*, area backend. Esempio di creazione di una scheda di dettaglio, tab 'Principale'

Cliccando il tab 'Attributi', apparirà una lista di tutti i tag disponibili, suddivisi secondo la configurazione dei gruppi e dei tag.

Figura 5.11 *JaLea*, area backend. Esempio di creazione di una scheda di dettaglio, tab 'Attributi'

Selezionando un tag, la sua descrizione e il relativo gruppo di appartenenza verranno visualizzati vicino all'elemento grammaticale da descrivere.

Figura 5.12. *JaLea*, area backend. Dettaglio del tab 'Attributi'. Si noti il nome del tag vicino all'elemento grammaticale selezionato

La creazione del testo per l'area di dettaglio avviene all'interno della sezione identificata dal tab relativo ('Dettaglio').

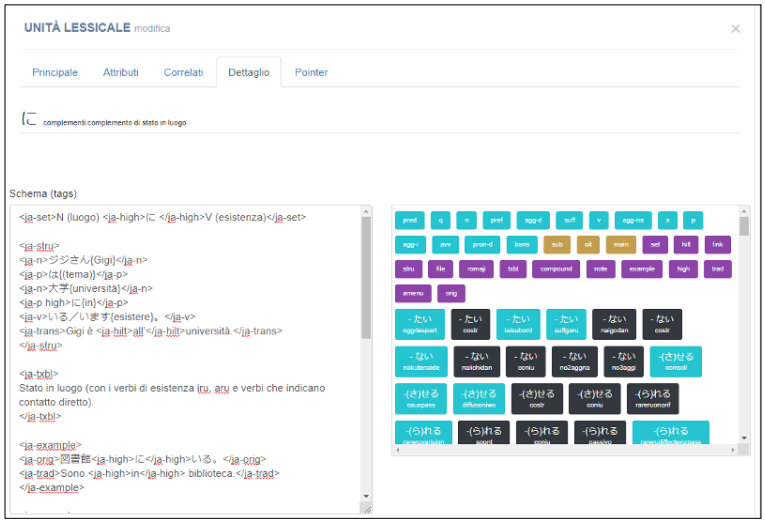


Figura 5.13 *JaLea* – area backend, tab 'Dettaglio'. Lo schermo è diviso in un area di testo per l'inserimento dei tag di *JaLea*, e un'area con i tag di *JaLea* selezionabili

La parte destra dello schermo contiene tutti i tag di analisi dei contenuti, definiti tramite ciascuna relativa funzionalità. I tag nella parte superiore sono quelli di sistema, riconoscibili perché sono più piccoli di quelli degli elementi grammaticali.

Mentre i tag di sistema permettono di creare tabelle, esempi, o note che appariranno nelle pagine di dettaglio, i tag grammaticali permettono di creare link automatici ad altre pagine di *JaLea*. Appoggiando il mouse su un tag ne appare la descrizione completa. Cliccando su uno di essi il codice HTML corrispondente appare nell'area di testo nella posizione del cursore.

Per produrre una pagina di dettaglio di *JaLea*, sarà sufficiente inserire il testo come segue. (Gli esempi seguenti sono tratti dalla pagina di backend riferita alla voce grammaticale に *ni*, relativa alle pagine di frontend indicate in ciascun link).

Creazione di un titolo

```
<ja-set>N (luogo) <ja-high>に</ja-high>V (esistenza)</ja-
set>
```

Il tag **<ja-set>** viene utilizzato per il titolo della pagina, il tag **<ja-high>**, per evidenziare l'elemento grammaticale oggetto della descrizione.

Creazione di una tabella di dettaglio

Esempio ジジさんは大学にいる／います *Gigi-san wa daigaku ni iru/imasu*

```
<ja-stru>
  <ja-n>ジジさん{Gigi}</ja-n>
  <ja-p>は{(tema)}</ja-p>
  <ja-n>大学{università}</ja-n>
  <ja-p high>に{in}</ja-p>
  <ja-v>いる／います{esistere}。</ja-v>
  <ja-trans>Gigi è <ja-hilt>all'</ja-hilt>università</ja-
  trans>
</ja-stru>
```

Il tag **<ja-stru>** identifica la tabella, il tag **<ja-n>** un nome, **<ja-p>** un pronome, **<ja-v>** un verbo.
Il tag **<ja-trans>** viene utilizzato per marcare il testo di traduzione di una frase.
Tra parentesi graffe è presente la trascrizione in italiano di ogni termine giapponese.

Creazione di un avviso

```
<ja-txbl>
  Stato in luogo (con i verbi di esistenza iru, aru e ver-
  bi che indicano contatto diretto).
</ja-txbl>
```

Il tag **<ja-txbl>** viene utilizzato per inserire un testo di avviso, un testo ovvero la cui lettura richieda particolare attenzione.

Creazione di esempi con la relativa traduzione in italiano

```
<ja-example>
  <ja-orig>図書館<ja-high>に</ja-high>いる。</ja-orig>
  <ja-trad>Sono <ja-high>in</ja-high> biblioteca</ja-trad>
</ja-example>
<ja-example>
  <ja-orig>
  道<ja-high>に</ja-high>木が倒れている。
  </ja-orig>
  <ja-trad>
  Un albero è [caduto] <ja-high>sulla</ja-high> strada
  </ja-trad>
  <ja-note>
    Il punto di contatto è 'la strada', direzione verso
    cui volge il cambiamento. L'uso della forma in <i>-te
    iru</i> è dovuto al mutamento (l'albero è caduto giù)
    e mantenimento della nuova condizione risultante (non
    si è mosso dalla strada).
  </ja-note>
</ja-example>
```

Questo codice viene interpretato e visualizzato nell'area di frontend di *JaLea* nel modo seguente:

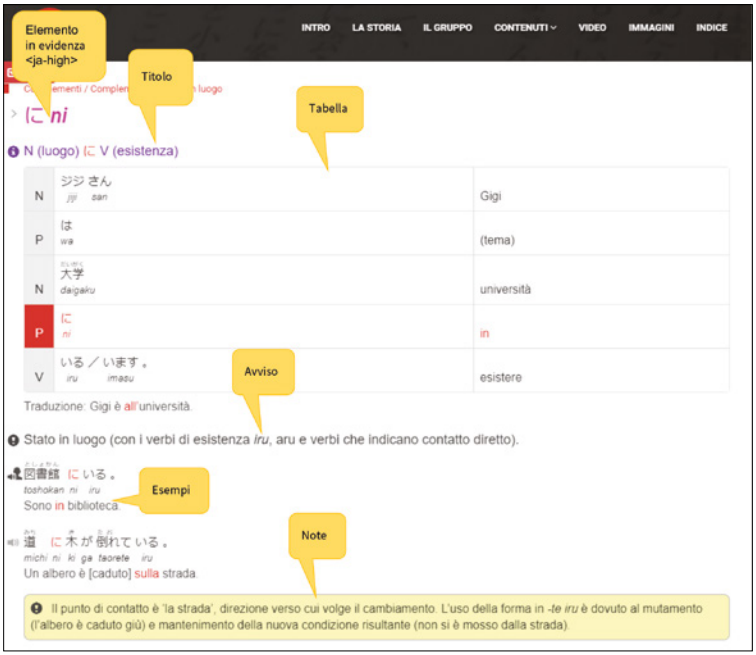


Figura 5.14 *JaLea*. Visualizzazione nell'area di frontend dei contenuti marcati dai tag (come da esempio) inseriti nell'area di backend

Si notino nella figura **5.14** anche le icone a sinistra di ogni frase di esempio che, se cliccate, permettono di riprodurre un file audio con la registrazione di un parlante madrelingua (esempio: *toshokan ni iru*) o l'audio derivante dalla conversione automatica del testo in voce, tramite un sistema automatico di *text-to-speech* (esempio: *michi ni ki ga taorete iru*). Le icone per la riproduzione di audio originale o convertito da testo sono differenti.

I tag HTML creati ad hoc per *JaLea* permettono pertanto di creare dei contenuti semplificandone per quanto possibile il processo di creazione. È da notare inoltre che il testo in giapponese viene inserito in giapponese, in *kana-kanji majiri*, e avviene trascritto automaticamente dal sistema di *JaLea* sia in *rōmaji* che in *hiragana*. Questo sistema verrà descritto nel dettaglio nel prossimo paragrafo.

Nel caso seguente, in figura **5.15**, invece, è stato utilizzato un tag grammaticale per creare un link tra la particella が (*ga*) presente nella frase di esempio, e la pagina di spiegazione relativa alla particella *ga* come marcatore del soggetto di una frase.

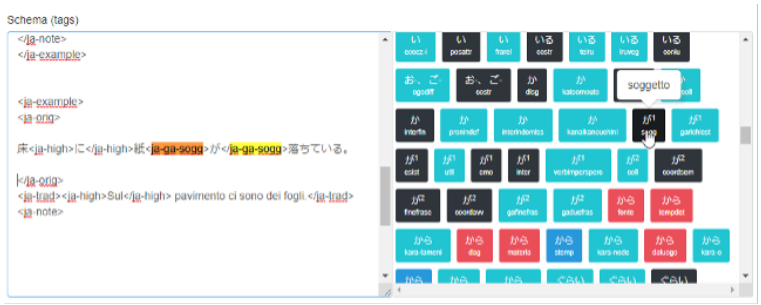


Figura 5.15 Nell'area di frontend viene creato un link automatico alla pagina di spiegazione grammaticale

Posizionando il cursore prima della particella が e cliccando il tasto [が sogg], la particella が viene racchiusa dal tag **<ja-ga-sogg>**. Nell'area di frontend, questo tag viene interpretato dal sistema, che crea un link automatico alla pagina di spiegazione della particella grammaticale **[fig. 5.16]**.

Figura 5.16 Nell'area di frontend viene creato un link automatico alla pagina di spiegazione grammaticale

5.3.4 Interface design

Come già descritto al § 4.3.5, a livello di progettazione degli elementi dell'interfaccia sono stati utilizzati elementi originali, non presenti nell'HTML standard, per permettere una interazione il più possibile fluida con l'utente-content manager. In particolare:

1. selettore di data: per definire la data di pubblicazione di una determinata scheda, oltre alla possibilità di inserire la data a mano è stato fornito uno strumento che permette la selezione della data da un calendario in pop-up come da schermata seguente.

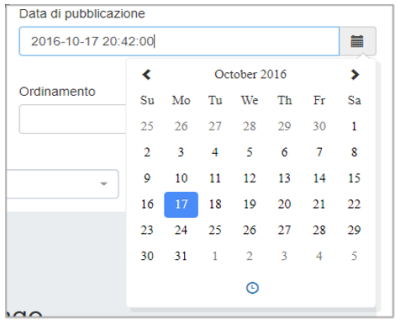


Figura 5.17 *JaLea*, area backend. Selettore di data

2. Al posto dei menu a tendina standard dell'HTML, sono stati implementati dei menu a tendina avanzati che permettono di filtrare gli elementi prima di selezionarli. Nello screenshot seguente, ad esempio, sono stati filtrati tutti i risultati per il morfema に (*ni*).

Figura 5.18 *JaLea*, area backend. Menu a tendina con filtro

3. Sono stati inseriti elementi selettori tipo *switch on-off* per rappresentare visivamente in modo chiaro il paradigma di attivazione/disattivazione di una determinata funzionalità.

Figura 5.19 *JaLea*, area backend. Selettore on/off

4. È stato implementato un gestore di file avanzato per aggiungere, modificare, eliminare i file di immagini e audio. Come da figura **5.20**, cliccando l'area denominata 'image', si apre una finestra pop-up con un menu con piccole icone (area superiore), e un'area dedicata ai contenuti presenti rappresentati sot-

to forma di icone più grandi (area inferiore). Attraverso le icone del menu è possibile caricare nuovi materiali, aggiungere nuove directory, cambiare il sistema di visualizzazione (icone o lista), filtrare i materiali per una chiave di ricerca. Cliccando l'icona di un'immagine, questa apparirà all'interno del riquadro 'image' e sarà assegnata alla scheda.

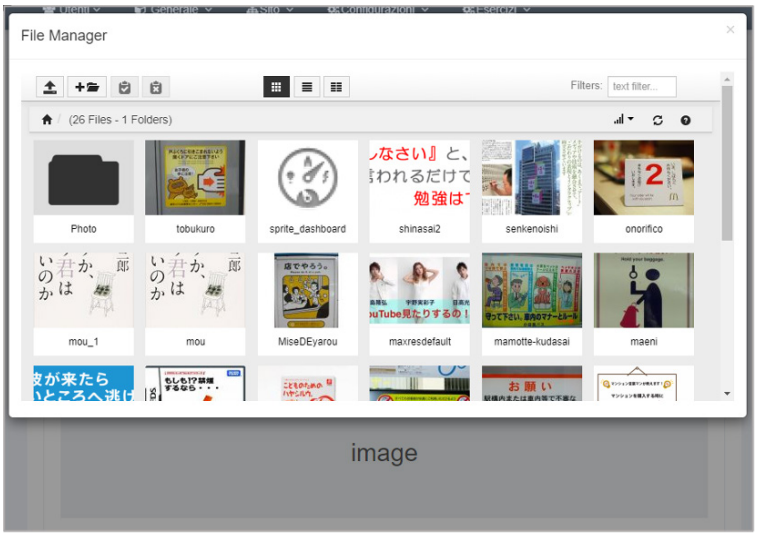


Figura 5.20 *JaLea*, area backend. Gestione file

5.3.5 Navigation Design

Il sistema di navigazione delle funzionalità di backend analizzate, in parte già stato descritto, segue un modello rappresentativo abbastanza comune per un processo CRUD, ovvero riproduce una tabella contenente numerosi record, ognuno dei quali se cliccato permette l'apertura di una area di dettaglio. Se l'elemento della tabella è sempre presente nella realizzazione pratica dei processi di CRUD, l'area di dettaglio può apparire o su pop-up modale,[4] o su una nuova pagina. È stato deciso di visualizzare le aree di dettaglio di *JaLea* in una nuova pagina, in quanto lavorare su un'area pop-up in schermi relativamente piccoli di smartphone e tablet potrebbe ri-

4 Il pop-up modale è una soluzione a cui si ricorre spesso nel design dei sistemi di interfaccia e navigazione attuali. Si tratta di visualizzare una nuova area con delle informazioni sopra il layout presente in quel momento. Tra il layout originale e la nuova area viene posizionato uno sfondo grigio trasparente che non permette l'accesso alle funzioni sottostanti e che indica visivamente la presenza di una nuova sezione.

sultare complicato. L'area di pop-up infatti comprende bordi e spaziature che riducono ancora di più l'area di lavoro utilizzabile. Inoltre l'uso eccessivo di 'modalità' in un software aumenta in genere il numero di passaggi mentali con la relativa diminuzione dell'associazione mentale tra affordance e funzionalità. Pertanto, in *JaLea* i pop-up modali sono limitati a richieste di conferma di operazioni da effettuare, e per attività relativamente veloci, come la ricerca di immagine, parole chiave, inserimento delle credenziali di accesso o scorciatoie di navigazione.

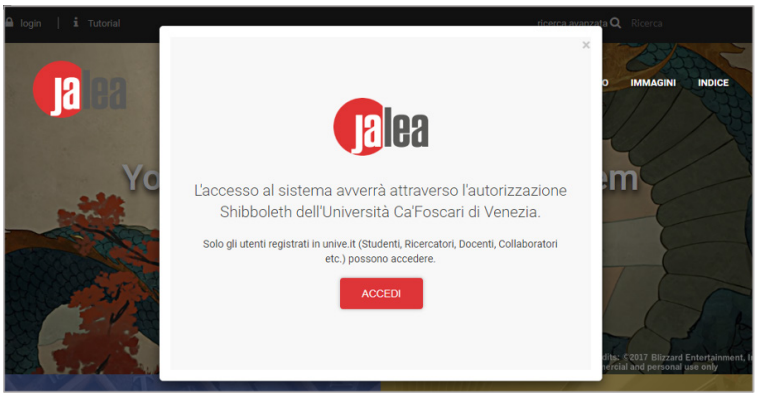


Figura 5.21 *JaLea*. Esempio di pop-up modale

5.3.6 Tema

Il 'Tema' corrisponde al livello di Surface Design definito da Garrett. Stabilisce lo stile grafico del prodotto e la gestione degli elementi nello spazio. Il tema utilizzato per *JaLea* è stato acquistato presso un rivenditore specializzato di terze parti. L'implementazione, pur richiedendo il tempo di adattamento del template al progetto, non ha necessitato la creazione degli stili base, di solito corrispondente a circa 1 mese/uomo di lavoro.

5.3.7 Feedback

Il processo di Feedback è avvenuto attraverso l'implementazione di una metodologia chiamata *double-loop learning* (Argyris 1977) concordata all'interno del team di lavoro. Questa metodologia, nata in ambito di organizzazione aziendale, permette la definizione di un circuito singolo (*single-loop*), ovvero un ciclo di 1) regole per l'azione; 2) verifica dei risultati, e 3) segnalazione di anomalie, insieme a un cir-

cuito doppio (*double-loop*) che implica un cambio del modello di riferimento e/o della metodologia di lavoro.

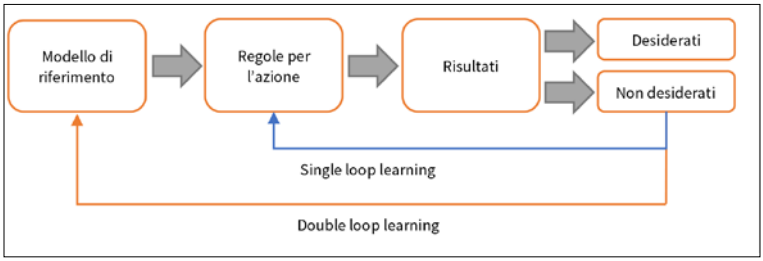


Figura 5.22 Argris 1977. Processo di single e double loop learning

Nel caso dello sviluppo in *JaLea* delle funzionalità descritte nelle pagine precedenti, si era concordato con il team di ricerca che la creazione dei tag, iniziata il 1° settembre 2016, sarebbe stata concordata di volta in volta con il team al completo, e che i tag decisi sarebbe poi stati inseriti dal system developer (Matelli).

Per 4 giorni (fino al 5 settembre 2016) il system developer ha provveduto alla creazione di ogni singolo tag concordato con il team, e del relativo stile grafico di presentazione se necessario. In questa prima fase di sviluppo si sono creati i tag 'complementi', 'subordinata finale + verbi di moto', 'subordinata concessiva', 'complemento di termine', 'predicato', 'nome', 'verbo', 'proposizione causale', 'complemento di causa', 'subordinata temporale', 'complemento di tempo determinato', 'aggettivi in i'.

Poiché lo sviluppo del sistema per la trasformazione del testo, tramite i tag, in testo visualizzabile dall'utente era ancora instabile, il content manager, quando utilizzava i tag concordati e riscontrava errori (blocco totale della pagina, mancata trasformazione del tag nell'hyperlink corrispondente ecc.), riportava immediatamente il problema allo sviluppatore, il quale provvedeva a risolvere il bug (single-loop).

Tuttavia, questo metodo non permetteva di lavorare adeguatamente, in quanto quando era necessario inserire un testo con regole grammaticali per le quali il tag corrispondente non era ancora stato creato, era necessario che il content manager interrompesse il lavoro per richiedere la creazione di un nuovo tag allo sviluppatore.

Dopo un'attenta riflessione sulle modalità di lavoro-feedback-correzione adottate fino a quel momento, il team di lavoro stabilisce che è necessario cambiare metodologia di sviluppo: il content manager potrà creare in autonomia i tag di cui necessita, il system developer si occuperà di risolvere bug, e creare uno stile grafico per i nuovi tag ove necessari (double-loop).

Questa nuova riorganizzazione delle attività del gruppo di ricerca e sviluppo ha permesso di agevolare l'attività di creazione dei tag e l'inserimento dei relativi contenuti. Dal 5 settembre 2018 fino al 13 ottobre 2018, sono stati creati infatti un totale di 309 tag, sufficienti per inserire gran parte dei contenuti attualmente presenti nella piattaforma *JaLea*.

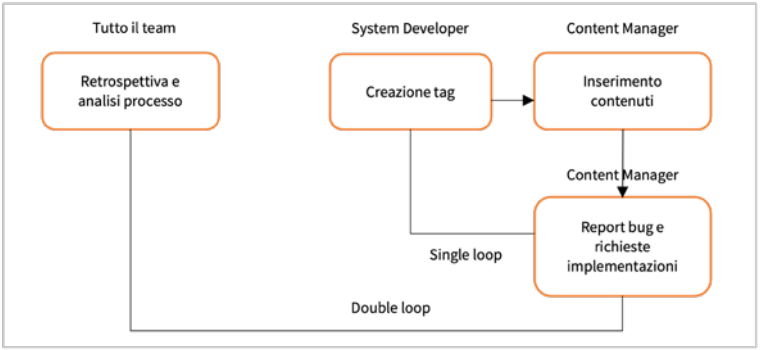


Figura 5.23 Flusso di lavoro prima dell'indagine retrospettiva

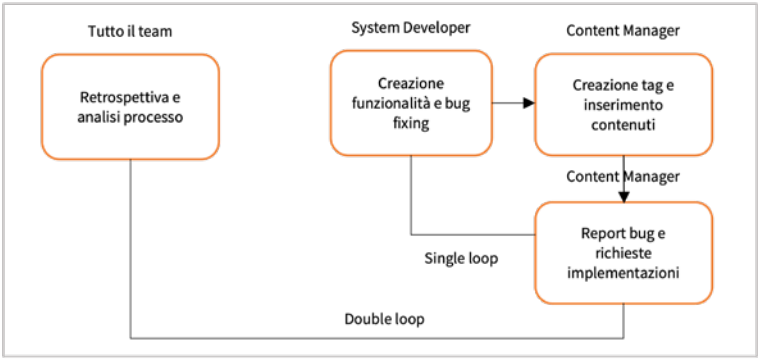


Figura 5.24 Flusso di lavoro dopo l'indagine retrospettiva

Come è stato evidenziato, attraverso il framework delineato è stato possibile descrivere un processo di sviluppo e i relativi particolari che permettono di rendere il prodotto utilizzabile in maniera ottimale dall'utente, nell'ottica di permettere una realizzazione continua di embodiment dell'utente con l'applicativo.

Un argomento che non è ancora stato affrontato e che merita particolare attenzione riguarda la presentazione del testo giapponese in *JaLea*. Al fine di agevolare l'inserimento, il testo giapponese viene

inserito in *kana-kanji majiri*, mentre *hiragana* (*furigana*) e *rōmaji*, inseriti in origine a mano, appaiono ora automaticamente grazie a un particolare algoritmo, analizzato nel prossimo paragrafo.

5.4 Applicazione del modello semplificato ADDIE-Garrett a *JaLea*: funzionalità di trascrizione e traslitterazione automatica

L'analisi nel presente paragrafo descrive le funzionalità di un algoritmo per cui è stata fatta richiesta di brevetto a Febbraio 2019, implementato in *JaLea* per la trascrizione in alfabeto sillabico *hiragana* e la traslitterazione in *rōmaji* della maggior parte dei testi. Attraverso la realizzazione in codice dell'algoritmo in esame, è possibile dividere un testo giapponese in unità lessicali lunghe LUW (*Long Unit Word*) e assegnare ad ogni unità la corretta trascrizione in *hiragana* e *rōmaji*.

In questo caso, il modello di analisi proposto è leggermente differente da quello dell'esempio precedente, per la presenza del livello denominato information architecture, necessario per spiegare l'algoritmo nei dettagli. L'attività di interpolazione del testo giapponese e la conseguente creazione di LUW avviene in questo caso in due tempi: il primo con i primi tentativi di implementazione, attraverso l'uso di un solo sistema di analisi morfologica del giapponese, e il secondo con la creazione dell'algoritmo e relativo programma realizzato in seguito all'analisi dei feedback degli utilizzatori.

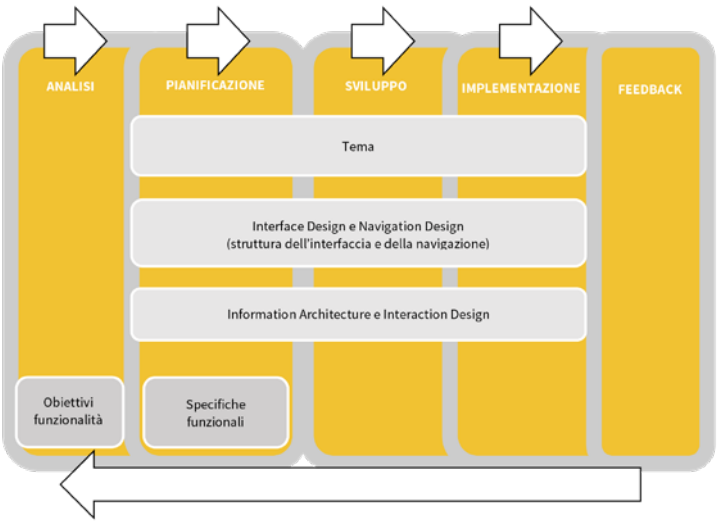


Figura 5.25 Modello semplificato ADDIE-Garrett per l'analisi della funzionalità della trascrizione automatica del testo giapponese

5.4.1 Analisi – Obiettivi della funzionalità (1/2)

La gestione dei contenuti in giapponese rappresenta un investimento di tempo considerevole se, per agevolare discenti con competenze linguistiche differenti, si decide di presentare ogni testo in giapponese anche con la relativa trascrizione in *furigana* e in *rōmaji*. In questo caso infatti è necessario introdurre il testo nelle tre differenti scritture (testo giapponese *kana-kanji majiri*, alfabeto sillabico *kana*, e caratteri latini *rōmaji*). È stato deciso pertanto di implementare in *JaLea* un sistema di trascrizione automatica che presenti automaticamente le tre tipologie, dando così la possibilità al content manager di concentrare il suo impegno sui soli contenuti in *kana-kanji majiri*.

Vista la complessità dell'operazione automatica di trascrizione, si è resa necessaria una completa analisi dello stato dell'arte, e dei vari test di utilizzo di programmi di analisi morfologica del testo giapponese, al fine di identificare una modalità di funzionamento implementabile in *JaLea*. Questi programmi, solitamente rilasciati gratuitamente, permettono, dato un testo giapponese in input, di dividerlo in morfemi e fornire per ognuno di questo, informazioni aggiuntive, quali la funzione grammaticale del morfema e, in alcuni casi, la trascrizione in *hiragana*. I primi prototipi di *JaLea* hanno visto pertanto l'utilizzo dei più noti programmi di divisione del testo giapponese in morfemi: *Juman* (Kurohashi, Kawahara 2006),[5] *Chasen* (Nara Institute of Science And Technology, 2007) e in seguito *MeCab* (Kudō 2006),[6] quest'ultimo usato anche come sistema di analisi morfologica alla base del software e-learning reading tutor (Kawamura 2012). Oltre a i suddetti programmi, è stato provato anche *Comainu*, software creato nel 2014 da Kozawa Shosuke (application engineer della compagnia Hatena[7] fino al 2017), che ha la particolarità di dividere il testo in LUW. Tuttavia, quest'ultimo software, oltre ad essere più lento degli altri sistemi di analisi morfologica, forniva spesso trascrizioni non corrette e pertanto è stato tralasciato. Dopo due mesi di test iniziati nel 2016, si è optato per il software *MeCab*, in quanto più veloce di *Juman* e *Chasen*, e in grado di fornire trascrizioni in *hiragana* in un formato output di dati più facile e più veloce da elaborare.

5 Sviluppato dal Dipartimento di Scienze e Tecnologia della Kyoto University: `http://nlp.ist.i.kyoto-u.ac.jp/EN/index.php?JUMAN`.

6 Sviluppato dall'Istituto di Scienze e Tecnologie di Nara nel 2006 e attualmente mantenuto da Taku Kodō

7 Per dettagli si veda la pagina della compagnia: `http://www.hatena.ne.jp`.

5.4.2 Specifiche funzionali (1/2)

Le prime specifiche funzionali relative alla funzionalità in esame sono state redatte procedendo come segue:

1. i contenuti testuali di *JaLea* vengono forniti a *MeCab*;
2. il risultato dell'analisi morfologica di *MeCab* viene analizzato da un programma in PHP che crea in *JaLea* un testo diviso in morfemi, ogni morfema è corredato di *furigana* e *rōmaji*;
3. attraverso una interfaccia sempre presente in fondo allo schermo **[fig. 5.26]** è possibile selezionare se visualizzare o meno le trascrizioni *hiragana* o *rōmaji* per i testi presenti;
4. attraverso un tasto della interfaccia del punto 3) è possibile attivare la funzionalità di dizionario. Posizionando il mouse su un carattere, il sistema visualizzerà un pop-up con la traduzione in italiano e un'animazione dei tratti di ogni singolo carattere[8] **[fig. 5.27]**.

5.4.3 Information architecture (1/2)

La prima versione del programma denominato *BunParser*, il cui algoritmo è oggetto di brevetto, non si occupava di creare unità lessicali lunghe, o corrette trascrizioni, ma forniva semplicemente il testo giapponese al programma *MeCab*, ne ricavava il risultato e lo visualizzava sullo schermo utilizzando tag HTML dedicati per le parti in *kana-kanji majiri*, *hiragana* e *rōmaji*.

Ad esempio, il seguente testo in giapponese: この木の実が食べられます。 viene analizzato e convertito da *MeCab* nel testo seguente.

```
この        連体詞,*,*,*,*,*,この,コノ,コノ
*木の実     名詞,一般,*,*,*,*,木の実,コノミ,コノミ
が          助詞,格助詞,一般,*,*,*,が,ガ,ガ
食べ        動詞,自立,*,*,一段,未然形,食べる,タベ,タベ
られ        動詞,接尾,*,*,一段,連用形,られる,ラレ,ラレ
ます        助動詞,*,*,*,特殊・マス,基本形,ます,マス,マス
。          記号,句点,*,*,*,*,。,。,。
```

Il programma *BunParser* analizza questo testo e crea un array associativo multidimensionale,[9] ovvero una struttura dati complessa, attraver-

8 Si veda per maggiori informazioni il § 4.5.1.

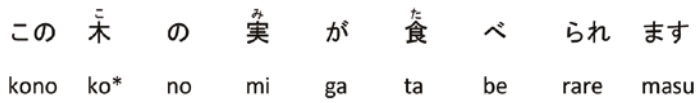
9 Un array associativo multidimensionale è una struttura dati a più livelli i cui elementi sono accessibili mediante nomi, quindi stringhe, anziché indici puramente numerici. Per dettagli si vedano le seguenti pagine di Wikipedia, «Array»: `https://it.wikipedia.org/wiki/Array`; «Array associativo»: `https://it.wikipedia.org/wiki/Array_associativo`.

so la quale memorizza i dati per un utilizzo successivo. Nella struttura dati (*array*) non vengono memorizzate solo le informazioni relative alla trascrizione, ma anche il risultato completo dell'analisi di *MeCab*, relativo anche alla funzione grammaticale di ogni singolo morfema. *BunParser* si occupa, inoltre, di convertire il testo da *hiragana* a *rōmaji*. La struttura dei dati dell'array risultante è come da schema seguente:

Tabella 5.1 Bunparser: risultato della conversione del testo generato da *MeCab* in array

[main] ⇒ この [tag] ⇒ 連体詞 [details] ⇒ [dettype] ⇒ [variant] ⇒ [cat] ⇒ [subcat] ⇒ [catform] ⇒ この [kana] ⇒ この [kata] ⇒ コノ [romaji] ⇒ kono	[main] ⇒ 木の実 * [tag] ⇒ 名詞 [details] ⇒ 一般 [dettype] ⇒ [variant] ⇒ [cat] ⇒ [subcat] ⇒ [catform] ⇒ 木の実 [kana] ⇒ このみ [kata] ⇒ コノミ * [romaji] ⇒ konomi *	[main] ⇒ が [tag] ⇒ 助詞 [details] ⇒ 格助詞 [dettype] ⇒ 一般 [variant] ⇒ [cat] ⇒ [subcat] ⇒ [catform] ⇒ が [kana] ⇒ が [kata] ⇒ ガ [romaji] ⇒ ga	[main] ⇒ 食べ [tag] ⇒ 動詞 [details] ⇒ 自立 [dettype] ⇒ [variant] ⇒ [cat] ⇒ 一段 [subcat] ⇒ 未然形 [catform] ⇒ 食べる [kana] ⇒ たべ [kata] ⇒ タベ [romaji] ⇒ tabe	[main] ⇒ られ [tag] ⇒ 動詞 [details] ⇒ 接尾 [dettype] ⇒ [variant] ⇒ [cat] ⇒ 一段 [subcat] ⇒ 基本形 [catform] ⇒ られ [kana] ⇒ られ [kata] ⇒ ラレ [romaji] ⇒ rare	[main] ⇒ ます [tag] ⇒ 助動詞 [details] ⇒ [dettype] ⇒ [variant] ⇒ [cat] ⇒ 特殊・マス [subcat] ⇒ 基本形 [catform] ⇒ ます [kana] ⇒ ます [kata] ⇒ マス [romaji] ⇒ masu

L'array può essere poi riconvertito all'occorrenza in testo, fornendo il risultato seguente.



Grazie alle funzionalità dell'interfaccia descritte nei paragrafi successivi, è possibile attivare o disattivare una o entrambe le trascrizioni e richiamare il dizionario online *a4edu* associato a *JaLea*, posizionando il mouse sopra un carattere.

5.4.4 Interaction Design

A livello di interaction design è stata predisposta un'interfaccia sempre presente nella parte inferiore dello schermo con tre tasti. Tasto dizionario, tasto *furigana*, tasto *rōmaji*.

La pressione del tasto *furigana* o *rōmaji*, come già detto, permette di attivare o disattivare una determinata trascrizione, mentre il tasto dizionario permette di visualizzare un pop-up con la traduzio-

ne del termine e l'animazione dei segni dei caratteri.

Il pop-up viene visualizzato tramite la tecnica AJAX, introdotta nel capitolo 5, pertanto l'attività è fluida e non richiede il caricamento della pagina rispettando il concetto di 'flow' (Csikszentmihalyi 2009) e di embodiment (Triberti, Brivio 2016).

5.4.5 Interface Design

Come si evidenzia in figura **5.26**, nella parte inferiore dello schermo è presente un'interfaccia con i tre tasti che rappresentano signifier per le trascrizioni e il dizionario. Premendo il tasto dizionario, si apre una piccola finestra pop-up e la relativa icona di attesa, prima della visualizzazione dei dati in modo da rispettare il principio introdotto nel capitolo 4 relativo alla percezione dei tempi d'attesa (Church et al. 1994, Scalar Timing Theory; Seow 2008).

Figura 5.26 *JaLea*. Interfaccia per visualizzare/nascondere le trascrizioni in *furigana* e *rōmaji* e attivare il dizionario

Figura 5.27 *JaLea*. Pop-up del dizionario

Come si vede dalla figura **5.27**, attivando la funzionalità 'dizionario' e posizionando il mouse sul termine 大丈夫 (*daijōbu*), appare pop-up di attesa prima del caricamento del dizionario. Il sistema non si blocca ma continua ad essere utilizzabile. Con la fine del caricamento, appare il dizionario e l'animazione che disegna i caratteri nell'ordine corretto.

Si è voluto utilizzare l'icona di una bandiera per indicare la traduzione in italiano del termine selezionato, in quanto in alcuni casi la traduzione in italiano non esiste ancora. I dati del dizionario infatti, vengono prelevati dall'applicazione Web *a4edu* (Mantelli, Mariotti, 2014) che si basa sulla libreria *JMDict* del gruppo di ricerca EDRDG e contiene al proprio interno 38.651 termini tradotti in italiano, grazie al lavoro del gruppo di ricerca del progetto *ITADICT* (Mariotti, Mantelli, 2011) che ha potuto contare sulla collaborazione di oltre 86 collaboratori fra studenti e docenti.[10]

5.4.6 Navigation Design

Tutte le attività cinestetiche relative a questa funzionalità non portano al caricamento di una nuova pagina, quindi la 'navigazione' verso pagine differenti in questo caso non avviene. È stato deciso di analizzare comunque anche questo elemento secondo il modello delineato ADDIE-Garrett in quanto il pop-up con il dizionario porta comunque ad una variazione nello stato della pagina, con la presenza di nuove informazioni ricavate attraverso un processo AJAX. In questo caso, l'utilizzo del pop-up di attesa **[fig. 5.27]** ha la funzione di indicare un cambiamento di stato, che comunica all'utente che il caricamento delle informazioni è in corso permettendo una migliore tolleranza dei tempi d'attesa.[11]

10 Si veda il § 1.3.6 per dettagli.

11 Si veda per maggiori informazioni il § 5.3.2.

5.4.7 Tema

Anche in questo caso l'ultimo livello del framework di Garrett (sensory design) è stato sostituito dal termine 'Tema' indicante il tema grafico utilizzato in *JaLea*.

5.4.8 Feedback

Attraverso l'analisi dei risultati generati da *BunParser*, e i feedback ottenuti tramite intervista qualitativa al content manager, al principal investigator e agli studenti che hanno testato il prototipo, si è capito che i testi giapponesi presenti in *JaLea* non erano esenti da problemi di trascrizione. I problemi principali erano relativi a 1) trascrizioni in *hiragana* non conformi al contesto d'uso più comune (esempio *Nipponjin* invece di *Nihonjin*, *ōita* invece di *tabun*), e 2) trascrizioni non corrette a causa della specificità dei software di analisi morfologica (*konokan* invece di *konoaida*). *MeCab*, infatti, dividendo prima il testo in morfemi (ad esempio: この間 in: a) この; b) 間), e poi effettuando l'operazione di trascrizione, attribuisce al *kanji* 間, fuori dal contesto originale, la trascrizione generica in *on-yomi*: *kan*. Il problema della traslitterazione non corrispondete diventa di difficile risoluzione, in quanto non necessariamente la trascrizione in *hiragana* dei singoli morfemi equivale alla trascrizione di una LUW, formata da più morfemi. Si prenda per esempio, il termine giapponese 一人 *hitori* (una persona): questo termine è formato da 2 morfemi 一 *ichi* (uno) e 人 *hito* (persona). Anche in questo caso, *MeCab*, divide innanzitutto il testo giapponese e poi applica la trascrizione in *hiragana* ad ogni morfema. La trascrizione fornita da *MeCab* in questo caso risulta quindi errata: **ichihito.*

La visualizzazione del testo suddiviso in morfemi sullo schermo, inoltre, era difficilmente accettabile, perché di difficile lettura: un verbo coniugato come 行きませんでした *ikimasen deshita* (non sono andato), ad esempio, veniva suddiviso dai plugin in un illeggibile: 行き|ませ|ん|で|した *iki-mase-n-de-shita*, oltre a costituire un ostacolo all'integrazione con le funzionalità del dizionario che non riconosceva i termini suddivisi in tal modo. Se '一人' *hitori* viene suddiviso in '一' e '人', posizionando il mouse sul primo carattere il sistema restituirà la traduzione italiana del termine 一 *ichi* 'uno', posizionandolo sul secondo, restituirà la traduzione di 人 *hito* 'uomo', ma non sarà possibile ottenere il riconoscimento del composto 一人 *hitori* e quindi la traduzione italiana corretta 'una persona'. Un ultimo problema che il prototipo *BunParser* ha dovuto affrontare era la gestione dei tag HTML, necessari per la struttura del testo di *JaLea* (come indicato nel § 4.4), normalmente eliminati però dal programma *MeCab*.

5.4.9 Analisi – Obiettivi funzionalità (2/2)

I problemi identificati nella fase di feedback, richiedevano la creazione di una funzionalità che, partendo dal risultato dell'analisi morfologica di *MeCab*, generasse un testo giapponese suddiviso in LUW di lunghezza assimilabile a quelle dei libri di testo di giapponese per principianti, e che producesse una trascrizione corretta di ogni LUW in *hiragana*, *katakana* e *rōmaji*, secondo le regole della grammatica giapponese.

5.4.10 Specifiche funzionali (2/2)

Per trascrivere correttamente le LUW corrispondenti all'unione di più morfemi, sono state necessarie 3 strategie: la prima riguarda la creazione di regole grammaticali da implementare nel codice, soprattutto per l'analisi delle quantità numeriche e dei contatori che le seguono, la cui trascrizione è influenzata da questi due elementi; la seconda è relativa alla creazione di un dizionario dedicato, *BunParserDic*, che presenti una nuova trascrizione laddove quella generata automaticamente dall'unione dei morfemi in LUW non sia corretta. Quest'ultima strategia è relativa alla possibilità di definire una particolare trascrizione per una o più LUW, solo per determinati testi e determinati contesti.

Con l'analisi dei feedback riportati da parte di docenti e studenti che utilizzavano *JaLea* e degli errori di trascrizione generati dall'analisi dei testi inseriti in *JaLea*, si è potuti giungere alla definizione delle seguenti specifiche funzionali, costituenti ogni passaggio del procedimento di analisi e identificazione dei termini:

a. il testo viene momentaneamente privato dei tag HTML se presenti, dei caratteri marcatori e dei relativi contenuti indicanti trascrizioni definite dall'utente;
b. il testo viene analizzato e suddiviso in morfemi con l'ausilio di *MeCab*;
c. ogni morfema viene esaminato tramite *BunParserDic*, e se necessario viene applicata la trascrizione corretta;
d. si verifica la presenza di trascrizioni definite da caratteri marcatori all'interno del testo, e viene memorizzata l'eventuale eccezione;
e. i tag HTML vengono riposizionati nella loro posizione originale;
f. si applicano strategie di trascrizione per i valori numerici, di solito strettamente legati a letture particolari;
g. si applicano strategie di trascrizione per i giorni del mese;

h. si uniscono i morfemi in LUW seguendo determinate regole specificate di proposito e descritte nel paragrafo seguente, in modo da generare la trascrizione in *rōmaji*;
i. ogni LUW viene esaminata tramite *BunParserDic*, e, se necessario, viene applicata la trascrizione corretta;
j. viene applicata l'eventuale trascrizione imposta tramite caratteri marcatori definita al punto d;
k. viene corretta la trascrizione in *rōmaji.*

5.4.11 Information architecture (2/2)

Le fasi a-k descritte nel paragrafo precedente si traducono a livello di information architecture nella creazione di una 'classe' formata da molteplici 'metodi',[12] ovvero di diversi costrutti che contengono numerosi blocchi di istruzioni, con variabili condivise tra ciascun blocco.

Seguendo la sequenza delle specifiche funzionali del paragrafo precedente, si indicano di seguito i metodi principali e la descrizione delle logiche utilizzate.

a. **Set**
Dopo avere istanziato la classe attraverso il comando: `$parser = new Bunparser()`, viene invocato il metodo **Set** con la seguente sintassi d'esempio:
`$result = $parser->set ("これは日本語のテストです")`
Il testo giapponese tra virgolette rappresenta l'input, mentre la variabile $result è l'output, ovvero un array associativo che contiene le LUW e le relative trascrizioni, come da § 5.4.2.
Grazie a questo metodo, il testo viene privato dai tag HTML, ma la posizione di ogni tag viene memorizzata, per poter essere ripristinata successivamente. Viene inoltre verificata la presenza dei caratteri marcatori %% e del relativo contenuto grazie alla stringa:
`%kanji1:trascrizione in hiragana1,kanji2:trascrizione in hiragana2%`
Questo tipo di struttura, all'interno del testo, permette di definire liberamente trascrizioni appropriate, ad esempio うまい *umai* (buono/delizioso) per 辛い *karai* (piccante), o per indicare trascrizioni diversificate per lo stesso *kanji* che appare più volte in una medesima frase. Ad esempio: la frase イタリアの方はイタリアの方へ帰りました *itaria no kata wa itaria no hō e ka-*

12 Nella programmazione orientata agli oggetti classi e metodi sono procedure o funzioni che permettono di descrivere oggetti caratterizzati dallo stesso insieme di comportamenti possibili. Wikiversità, *Programmazione orientata agli oggetti* [online]: `https://it.wikiversity.org/wiki/Programmazione_orientata_agli_oggetti`.

erimashita (Un signore italiano è tornato verso l'Italia), contiene due volte il *kanji* 方, con due trascrizioni e significati differenti: *kata* (signore) *e hō* (direzione). Attraverso l'utilizzo dei caratteri marcatori %% è possibile specificare in questo caso le due trascrizioni diverse fra loro: イタリアの方はイタリアの方へ帰りました%方:かた,方:ほう%. Pertanto, se nel testo è presente questa casistica, i dati relativi alle trascrizioni inserite vengono memorizzati, mentre i caratteri marcatori e il relativo contenuto sono eliminati.

b. **JImport**
Il testo giapponese è analizzato e convertito in morfemi tramite *MeCab*, mentre il risultato viene salvato in un array associativo.

c. **applyDictionary**
Questo metodo si occupa di consultare le informazioni di *BunParserDic* e verificare una possibile corrispondenza tra ogni morfema e il dizionario delle regole
La struttura delle informazioni di *BunParserDic* è quella di un vasto array associativo, il cui formato è il seguente:

```
[
"chiave_entrata1 = valore_entrata1, chiave_entrata2 =
valore_entrata2" ⇒ "chiave1 = valore1, chiave2 = va-
lore2"
]
```

Ad esempio, tramite il record:

```
["kana = は, details = 係助詞 " ⇒ "romaji = wa"]
```

si definisce che per ogni chiave 'kana' dell'array il cui valore è は *ha/wa*, e la cui chiave 'details' è 係助詞 *kakarijōshi* (particella di collegamento), la trascrizione in *rōmaji* dev'essere *wa* e non *ha*. Come già indicato nel § 5.4.2, infatti, nell'array multidimensionale vengono memorizzate tutte le informazioni fornite da *MeCab*, compresa la funzione grammaticale di ogni singolo morfema.
Si presentano di seguito alcuni esempi:

```
["main = 明日, tag = 名詞" ⇒ "kana = あした, romaji =
ashita"]
```

In questo caso, per ogni elemento la cui chiave 'main' ha il valore 明日 *ashita* e la cui chiave 'tag' (funzione grammaticale)

ha il valore 名詞 *meishi*, il valore della chiave 'kana' diventa あした e la relativa traslitterazione in caratteri alfabetici: *ashita*.

```
["kana = を, tag = 助詞" ⇒ "romaji = o" ]
```

Per ogni chiave 'kana' dell'array il cui valore è を *wo/o*, si richiede la traslitterazione in caratteri alfabetici *o.* In questo modo si forza quindi la traslitterazione del carattere を in *o,* seguendo lo standard di traslitterazione Hepburn.

d. **applyUserDictionary**
Questo metodo permette di applicare le eventuali trascrizioni dichiarate tra i caratteri marcatori %% memorizzate tramite il metodo 'set', ad esempio 方 come ほう *hō* o かた *kata*, a seconda delle necessità del contesto.

e. **appendTags**
Vengono aggiunti nuovamente i tag HTML tolti prima di effettuare l'analisi e la conversione del testo tramite *MeCab*.

f. **recursiveFixNumbers, numbersReading, lastNumRules, CounterReading**
Attraverso questi metodi:
1. tutte le cifre numeriche consecutive vengono accorpate in un'unica LUW;
2. vengono create le trascrizioni corrette per le quantità numeriche, ad esempio 1.390: せんさんびゃくきゅうじゅう *sensanbyakukyūjū* (12.696), いちまんにせんろっぴゃくきゅうじゅうろく *ichimannisenroppyakukyūjūroku*, che altrimenti, tramite *MeCab* risulterebbero せんさんきゅうぜろ *sensankyūzero* e いちにろくきゅうろく *ichinirokukyūroku*;
3. vengono create le trascrizioni corrette per i contatori, considerando la parte numerica che li precede. Ad esempio: se l'ultima cifra del numero che precede il contatore 分 *fun* 'minuto' è 1,3,6,8 o 0, si applica la trascrizione ぷん *pun*. Nel caso del contatore 本 *hon* (libro), invece, si applica la trascrizione ぽん *pon* solo nel caso in cui l'ultima cifra del numero che lo precede sia 1, 6, 8, e la trascrizione ぼん *bon* nel caso la cifra che lo precede sia 3. *BunParser* contempla tutti i possibili contatori.

g. **daysReading**
Nel caso dei giorni del mese, *MeCab* interpreta ogni singolo carattere come un singolo morfema. Pertanto non è in grado di identificare la trascrizione corretta dei giorni, che si basa sull'analisi completa del termine (formato da più *kanji*), e non sul singolo carattere che lo compone. Questo metodo si

occupa di formare una LUW completa del giorno in esame, unendo insieme i vari elementi (cifra e contatore) dell'array. Ricrea quindi le trascrizioni corrette basandosi sulle regole della grammatica giapponese, secondo i casi seguenti:

- 一日 (oppure 1日) **MeCab*:*ichinichi* 'un giorno';
  se il carattere che segue non contiene il classificatore 目, viene trascritto come ついたち *tsuitachi* (il primo del mese).
- 二日 (oppure 2日) **MeCab*:*ninichi*;
  viene correttamente trascritto in ふつか *futsuka* (due giorni / il due del mese).
- 三日 (oppure 3日) **MeCab*:*san'nichi*;
  viene correttamente trascritto in みっか *mikka* (tre giorni / il tre del mese).
- 四日 (oppure 4日) **MeCab*:*yon'nichi*;
  viene correttamente trascritto in よっか *yokka* (quattro giorni / il quattro del mese).
- 五日 (oppure 5日) **MeCab*:*gonichi*;
  viene correttamente trascritto in いつか *itsuka* (cinque giorni / il cinque del mese).
- 六日 (oppure 6日) **MeCab*:*rokunichi*;
  viene correttamente trascritto in むいか *muika* (sei giorni / il sei del mese).
- 七日 (oppure 7日) **MeCab*:*nananichi*;
  viene correttamente trascritto in なのか *nanoka* (sette giorni / il sette del mese).
- 八日 (oppure 8日) **MeCab*:*hachinichi*;
  viene correttamente trascritto in ようか *yōka* (otto giorni / l'otto del mese).
- 九日 (oppure 9日) **MeCab*:*kyūnichi*;
  viene correttamente trascritto in ここのか *kokonoka* (nove giorni / il nove del mese).
- 十日 (oppure 10日) **MeCab*:*jūnichi / ichi zero nichi*;
  viene correttamente trascritto in とおか *tooka* (dieci giorni / il dieci del mese).
- 十四日 (oppure 14日) **MeCab*:*jūyon'nichi / ichi yon nichi*;
  viene correttamente trascritto in じゅうよっか *jūyokka* (quattordici giorni / il quattordici del mese).
- 二十日 (oppure 20日) **MeCab*:*nijūnichi / ni zero nichi*;
  viene correttamente trascritto in はつか *hatsuka* (venti giorni / il venti del mese).

h. **postProductionRules**

BunParser utilizza questo metodo per creare LUW partendo dai morfemi creati da *MeCab*.
Attualmente le regole implementate per la creazione di LUW sono le seguenti:

- verbo seguito dall' ausiliare negativo *-zu*, o dalla forma passata, ad esempio:
  忘れ　ず → 忘れず *wasurezu* ぶつか　った → ぶつかった *butsukatta*;
- parte del verbo e suo ausiliare (*-reru*, *-rareru*), ad esempio:
  はさま　れる → はさまれる *hasamareru*;
- verbo seguito da ausiliare di collegamento (almeno che non si tratti delle particelle から *kara e* ので *node* 'perché'), ad esempio:
  歩い　て → あるいて *aruite*;
- だ *da/* です *desu* seguito da ausiliare *aru*, ad esempio:
  で　ある *aru* → である *dearu*;
- aggettivi di tipo coniugabile al passato, ad esempio:
  かわいか　った → かわいかった *kawaikatta*;
- aggettivi e sostantivi di appartenenza geografica dove veniva spezzata la radice e il suffisso aggettivale, ad esempio:
  アメリカ　人 → アメリカ人 *amerikajin* 韓国　人 → 韓国人 *kankokujin*;
  verbi che esprimono incertezza e probabilità quali でしょう *deshō* e だろう *darō*;
- nomi composti, ad esempio:
  図書　館 → 図書館 *toshokan*イタリア　語 → イタリア語 *itariago* 誕生　日 → 誕生日 *tanjōbi*;
- alcuni casi particolari non gestiti correttamente da *MeCab* quali ad esempio alcune interiezioni, ad esempio:
  初め　まして → 初めまして *hajimemashite* かも　しれ　ません → かもしれません *kamoshiremasen*.

i. **applyDictionary**
Alle LUW create viene nuovamente applicato il dizionario *BunParserDic* per un'ulteriore verifica.

j. **applyUserDictionary**
Viene riutilizzato questo metodo per applicare le eventuali trascrizioni dichiarate tra i caratteri marcatori %%

k. **fixRomaji**

Questo metodo si occupa di risolvere i problemi di traslitterazione in caratteri latini che avvengono quando BunParser crea LUW la cui trascrizione in *hiragana* termina con il carattere di raddoppiamento consonantico っ *chīsai tsu*. Nel caso del sintagma: 60本 *rokujuppon* (60 bottiglie), ad esempio, *BunParser* accorpa prima la parte numerica in un unico LUW, e opera quindi sulla trascrizione di 60 e 本 che è rispettivamente ろくじゅっ *rokujup* (sessant) ぽん *pon* (ta bottiglie). L'operazione di traslitterazione in caratte-

ri latini però non è in grado di interpretare autonomamente il carattere っ *tsu*, pertanto quest'ultimo viene traslitterato in **roku-juっpon*. Il metodo in questione, quindi, si occupa di verificare la presenza del carattere っ *tsu* all'interno della stringa in caratteri latini, ed eventualmente di correggere la trascrizione.

5.4.12 Esempi di comparazione tra MeCab e BunParser

In questo paragrafo vengono proposti tre esempi il cui testo è stato analizzato ed elaborato sia da *MeCab* che da *BunParser*; compilando una stringa in giapponese da sottoporre al metodo 'set' (cf. § 5.4.10), il risultato non sarà una stringa alfanumerica, ma un array associativo che contiene diversi tipi di dati. Di seguito si ipotizza che questo array venga poi ritrasformato in un testo.

Esempio 1: quantità numeriche

この本は普段３００００円ですので、１５０００円では買えません
Kono hon wa fudan san man en desu node, ichi man go sen en de wa kaemasen
(Siccome questo libro di solito costa 30.000 yen, non si può comprare per 15.000 yen)

MeCab
この|本|は|普段|３|０|０|０|０|円|です|ので|１|5|０|０|０円|で|は|買え|ませ|ん*Kono hon ha fudan* ***san zero zero zero zero*** *en desu node* ***ichi go zero zero zero*** *en* de wa kaemasen

BunParser
この|本|は|普段|３００００円|です|ので|１５０００円|で|は|買えません
Kono hon wa fudan ***san man en*** *desu node* ***ichi man go sen*** *en de wa kaemasen*

Dal punto di vista della divisione in SUW/LUW (*Short Unit Word*/*Long Unit Word*), MeCab tratta ogni singola cifra come una SUW indipendente, pertanto non riesce a generare una trascrizione corretta per le stringhe numeriche. Inoltre, il verbo alla forma potenziale cortese negativa 買えません *kaemasen* (non si può comprare), viene suddiviso in tre SUW: 買え|ま|せん *kae-ma-sen*.

BunParser, basandosi su dizionari appositamente creati, genera correttamente le trascrizioni per le componenti numeriche, e visualizza il verbo coniugato come con una LUW 買えません *kaemasen*, risultato dell'unione delle tre SUW iniziali *kae-ma-sen*.

Esempio 2: LUW

私はこの間、昔からの韓国人の友達に会って、レストランで一緒にビールを三本も飲みました
watashi wa, konoaida, mukashi kara no kankokujin no tomodachi ni atte, resutoran de issho ni bīru o san bon mo nomimashita.
(Io recentemente ho incontrato un vecchio amico coreano e abbiamo bevuto insieme 3 birre al ristorante).

MeCab
私|は|この間　、昔|から|の|韓国|人|の|友達|に|会っ|て、レストラン|で|一緒|に|ビール|を　三|本|も|飲み|まし|た
*Watashi ha kono**kan** mukashi kara no kankoku jin no tomodachi ni at te, resutoran de issho ni bīru wo san **hon** mo nomi mashi ta.*

BunParser
私|は|この間　、昔|から|の|韓国人|の|友達|に|会って、レストラン|で|一緒|に|ビール|を|三本|も|飲みました
*Watashi wa kono**aida** mukashi kara no **kankokujin** no tomodachi ni **atte** resutoran de issho ni bīru o san**bon** mo **nomimashita.***

In questo caso, rispetto a *MeCab*, si può notare la corretta trascrizione per この間 *konoaida* (recentemente) invece delle errate **konokan*, 三本 *sanbon* (tre bottiglie) e non **sanhon*
Altre LUW proposte sono:
韓国　人 → 韓国人 *kankokujin* (coreano)
会　って → 会って *atte* (incontrato)
飲み　ました → 飲みました *nomimashita* (avere bevuto)

Esempio 3: contatori

僕の誕生日は2月10日です。4時30分にパーティをやります。
Boku no tanjōbi wa 2 gatsu tōka desu. yoji sanjuppun ni pāti o yarimasu.
(Il mio compleanno è il 10 febbraio. Faccio la festa alle 4:30)

MeCab
僕|の|誕生|日|は|2月|1|0|日|です。4|時|3|0|分|に|パーティ|を|やり|ます。
*Boku no **tanjō bi** ha 2 gatsu **ichi zero nichi** desu. **yon ji san zero fun** ni pāti wo **yari masu***

BunParser
僕|の|誕生日|は|2月|10日|です。4時|30分|に|パーティ|を|やります。
*Boku no **tanjōbi wa** 2 gatsu **tōka** desu. **yoji** sanjuppun ni pāti o yarimasu.*

BunParser, grazie allo specifico dizionario incorporato, consente trascrizioni corrette per 10日 *tōka* (il giorno 10), 4時 *yoji* (le quattro), 30分 *sanjuppun* (e 30 minuti), laddove *MeCab*, lavorando sui singoli morfemi, non è in grado di farlo.

In questo ultimo caso vengono create le seguenti LUW:
誕生 日 → 誕生日 *tanjōbi* (compleanno)
１０　日 → １０日 *tōka* (il giorno 10),
４　時 → 4時 *yoji* (le ore 4)
３０　分 → ３０分 *sanjuppun* (i minuti 30)
やり　ます → やります *yarimasu* (faccio)

5.4.13 Osservazioni

Nella presente monografia, l'applicazione del modello semplificato ADDIE-Garrett a *BunParser* ha fini puramente dimostrativi. Si intende infatti dimostrare la validità di questo modello per la progettazione e l'analisi di determinate funzionalità in un'ottica di experience design. Tuttavia, nella implementazione pratica in *JaLea*, l'analisi dei dati raccolti dopo una successiva fase di feedback ha permesso un ulteriore miglioramento di *BunParser*.

Nell'implementazione attuale di *JaLea*, infatti, la libreria è sufficientemente evoluta da fornire al dizionario anche informazioni di tipo grammaticale, quali ad esempio i tempi verbali, in modo da ottenere informazioni ulteriori nell'area di pop-up di traduzione come si vede da immagine seguente.

Figura 5.28 *JaLea*. Indicazione del tempo verbale nell'area di pop-up del dizionario

5.5 Considerazioni sul modello semplificato ADDIE-Garrett

Normalmente ADDIE è caratterizzato da una certa rigidità metodologica negli step necessari al workflow di sviluppo, ma proprio per il fatto che il flusso di lavoro è caratterizzato da una struttura definita è possibile creare un modello universale sul quale sovrapporre i differenti livelli di astrazione del framework di Garrett (2007). Una volta che questo modello è completo delle sue componenti diacroniche (ADDIE) e astrattive (Garrett), è possibile modularlo (modello semplificato) secondo la funzionalità da analizzare e implementare selezionando gli elementi del modello più adeguati. Tuttavia, non necessariamente il percorso di sviluppo deve seguire gli step ADDIE in modo sequenziale. Il modello, infatti ha lo scopo principale di indicare quali elementi di user experience debbano essere analizzati nelle varie fasi di sviluppo, ma a seconda della cultura organizzativa è possibile anche ottimizzare il processo di pianificazione, produzione e feedback attraverso metodologie di tipo *Agile*. Queste metodologie permettono di incentivare la comunicazione dei team di lavoro e di rispondere più velocemente alle eventuali ripianificazione dei processi di sviluppo. Tuttavia è necessario che il team sia a conoscenza di come funzionano nei dettagli queste metodologie e che abbia una notevole autonomia nei processi decisionali, affinché possano essere efficaci. Come suggerito da Cao, Mohan e Xu (2009, 332): «such methodologies need to be adapted to suit the needs of different contexts».

Il vantaggio del modello semplificato ADDIE-Garrett qui sottoposto risiede quindi nella sua adattabilità.

Non solo, infatti, si può conformare a differenti situazioni e contesti nell'analisi degli oggetti complessi, ma può essere applicato anche a tutti gli artefatti digitali sufficientemente complessi da necessitare un'analisi stratificata dei processi sottostanti le singole funzionalità, e non necessariamente limitandone l'utilizzo al software, ma suddividendolo in due moduli, sia a livello diacronico che sincronico. Ad esempio, se si intende evidenziare le evoluzioni del processo di sviluppo dell'interfaccia di un applicativo, può essere sufficiente analizzare la fase di sviluppo e feedback del livello diacronico e dell'interfaccia e delle metodologie di navigazione a livello sincronico. Al contrario, se lo sviluppo dell'applicativo ha considerato la progettazione ad hoc della grafica dell'interfaccia, potrebbe essere utile descrivere tutte le strategie utilizzate a livello di sensory design, come descritto dal modello di Garrett, soprattutto nel caso di feedback non particolarmente positivi sull'aspetto grafico dell'applicativo da parte degli utenti.

La stessa funzionalità può essere analizzata modificando il modello ai fini di identificarne eventuali criticità insorte durante il processo di feedback. Se ad esempio il processo di feedback indica problemi nella fase di utilizzo dell'interfaccia di un applicativo, nel ripartire dalla fase di analisi, potrebbe essere utile aumentare il dettaglio d'a-

nalisi della fase sincronica, considerando aspetti relativi ai processi di infrastruttura e programmazione, magari non sufficientemente presi in considerazione durante la prima fase di progettazione.