POP-ODE manual

Vincenzo Lombardo(✉) and Rossana Damiano

CIRMA and Dipartimento di Informatica, Università di Torino, Torino, Italy
vincenzo.lombardo@unito.it

Abstract. This document describes how to use the POP-ODE toolsuite for the annotation of a drama according to ontology Drammar tenets. We go through the various phases, assuming that the application is local to the annotator machine.

1 Introduction

Drama annotation is the process of annotating the metadata of a drama. Initiative POP-ODE (POPulating Ontology Drama Encodings) relies on a web–based system, that provides a friendly interface to the dramatic qualities of a text. The metadata are defined through an annotation schema that descends from a formal theory of drama, Drammar, expressed through a computational ontology. The Drammar approach for the annotation of content metadata of dramatic texts, which relies on Semantic Web technologies, has been described in a number of papers along with its application to a few tasks: informing models of automatic storytelling on the calculation of emotions [2]; supporting the didactics of drama through the visualization of schematic intention charts [5]; preserving drama as an intangible form of cultural heritage [4]; encoding Stanislavsky's action analysis for actor rehearsals [1]; providing a clear status to annotation documents [3].

2 The POP-ODE pipeline

POP-ODE consists of a pipeline and a number of tools for the accomplishment of the annotation task of metadata for dramatic texts (see Figure 1). The Figure 1 shows the pipeline vertically, on the left, from top to bottom. A drama encoding annotator (at the top, left) works through a web-based interface to fill the tables of a data base built according to the tenets of ontology Drammar, which encodes its elements, namely story units, characters, actions, intentions or plans, goals, conflicts, values at stake (emotions are calculated automatically from these data). At the same time, the annotator can select the text chunks that correspond to some annotation (from the .txt file), which is visible in the web interface.

The ontology axioms are encoded (through the well-known Protègè editor[1]) in a conceptual model, OWL file. The mapper module DB2OWL converts the

[1] `http://protege.stanford.edu`, visited on 21 July 2017.

data base tables into an OWL file, actually a Drammar Instantiated Ontology file (OWL DIO file). A further software module, OWL2CHART, extracts the individuals and properties in a XML Drammar Chart file, which is then visualized by the interactive chart module, the user can interact with [5]. The interactive chart module, developed for visualization purposes and as a teaching device, here supports the validation of the produced ontology, allowing for a fast checking of the encoded axioms. On right column, the figure shows examples from each step. The top of the column shows a thumbnail of the web interface (see detail below). As example for the data base, there are two example tables, Agent and Value, connected through the Agent identifier (Hamlet has honesty as a value). The assertion example from the DIO OWL file concerns the agent Marcellus who intends the plan of reaching the guard post. The example from the XML chart representation shows the attributes of the plan above, which determine its visualization color and shape in the interactive chart.

In the next sections, we describe the use of the software packages that implement the POP-ODE pipeline described here.

3 Preliminaries: installing the dynamic web server on the machine

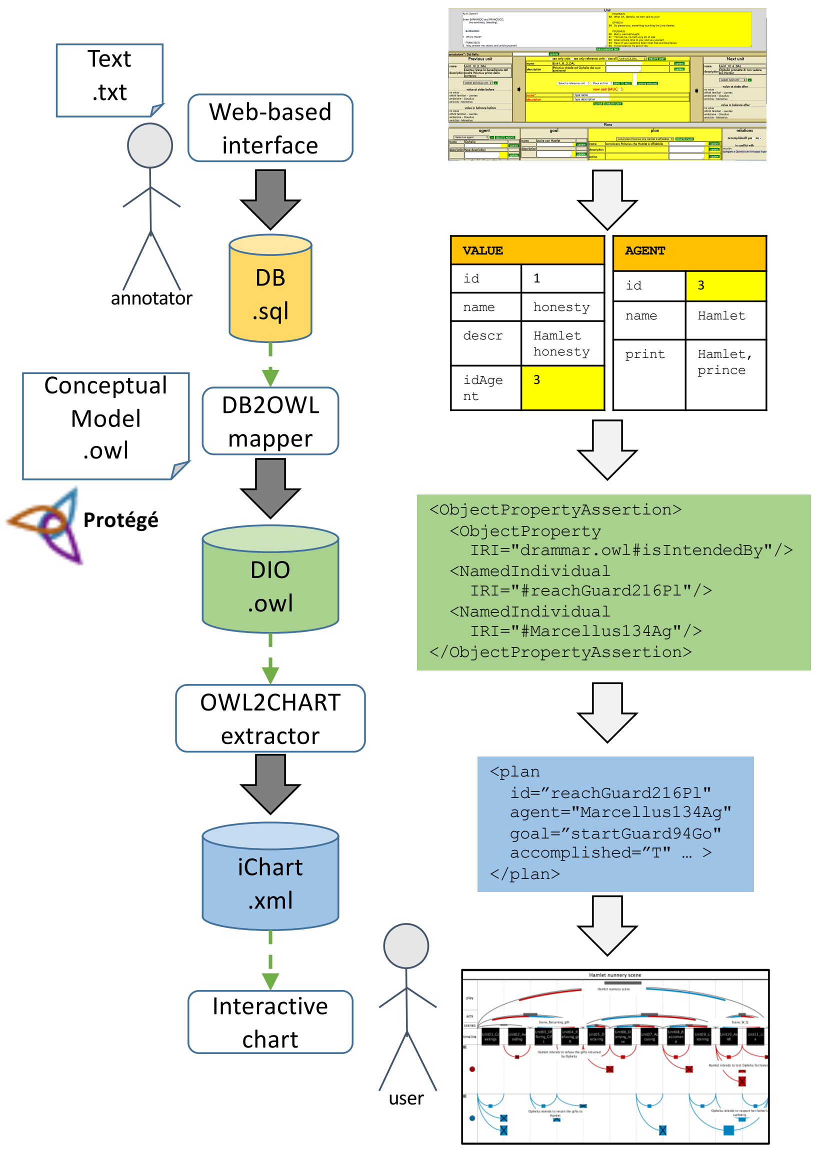
Here we describe how to install the dynamic web interface on your own machine. All our examples are referred to the MAMP solution for Mac users; however, the same works for other platforms as well as some remote server (e.g., altervista.org).

First, check whether you have a solution to run dynamic web sites on your computer. In the case of Mac, MAMP is a commodity already installed on the machine. MAMP is a solution stack composed of free and open-source and proprietary commercial software used to run dynamic web sites on Mac OS computers (from Wikipedia entry for MAMP). MAMP is an acronym of macOS, the operating system; Apache, the web server; MySQL, the database management system; and PHP, Perl, or Python, all programming languages used for web development.

Inside the $MAMP$ folder (within the *Applications* folder), we can find folder *htdocs*. Copy the $DAWI$*.zip* file within the htdocs folder; uncompress it (folder DAWI). Change the name of the folder to your drama (e.g., $HamletAnnotation$). Launch MAMP. Open the web start page from the MAMP interface or, in general, access the *phpMyAdmin* page from MAMP home page[2]. Create a new data base with the name of your current project (e.g., $HamletDB$) and import the *sql* file in the DAWI folder ($DAWI$*.sql*). Before proceeding, open the file *connessione.php* within folder $DAWI/annotazione/include$, and set the parameters as appropriate for your machine. Also, within folder $DAWI/annotazione/Texts$, insert the appropriate *sample.txt* file.

In another tab of your opened browser, load the URL `http://localhost:8888`. The page shows the list of your annotation projects. Click on one project

[2] `http://localhost:8888/MAMP/`



VALUE	
id	1
name	honesty
descr	Hamlet honesty
idAgent	3

AGENT	
id	3
name	Hamlet
print	Hamlet, prince

```
<ObjectPropertyAssertion>
  <ObjectProperty
    IRI="drammar.owl#isIntendedBy"/>
  <NamedIndividual
    IRI="#reachGuard216Pl"/>
  <NamedIndividual
    IRI="#Marcellus134Ag"/>
</ObjectPropertyAssertion>
```

```
<plan
  id=”reachGuard216Pl"
  agent="Marcellus134Ag"
  goal=”startGuard94Go"
  accomplished=”T" … >
</plan>
```

Fig. 1. The ODE-Pop annotation pipeline. The lower part shows examples of the

(e.g., *HamletAnnotation*), to start your work. As you work on your annotation, the data base fills with the data you insert.

4 POP-ODE web interface

Figure 2 shows the web interface for the annotation.

4.1 Description

The top of the figure shows the text selector: on the left, the text from some authoritative source (e.g., Hamlet from Shakespeare's navigators), on the right, the text chunk that pertains to the unit selected below. The middle of the figure shows the unit annotation, that is the actions that have been identified by the annotator in the selected segment of the text, recognized as a bounded unit. On the left and the right of the unit annotation are the previous and the following unit in the story timeline, with the values that are at stake or at balance before and after the current unit.

4.2 Usage

To start any activity, the user must select or create a unit (inserting its name, obligatory, and description), together with the annotator name (in case of a creation). Units can be minimal (or $reference$ unit, in the interface) or non minimal (i.e., composed of many reference units): in case of non minimal, the reference units that compose it can be selected and ordered within the interface. The user can also select the previous and the next units for the reference units.

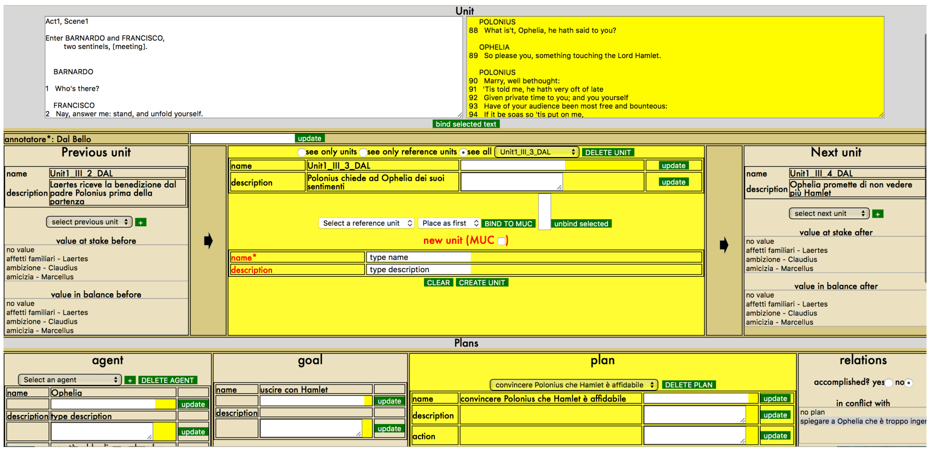
The bottom of the figure concerns the plans that motivate such a unit. The user must select or create a plan, by inserting a name (obligatory), a description, and an action (the latter describes the actual action carried out to implement the intention committed to). On the left columns, the user can also set a goal for this plan (facultative) and an agent (obligatory). The agent is described by a name (obligatory), a description, a number of values, an attribute of its pleasantness, and a number of agents or objects he/she likes or dislikes. Finally, on the right of the yellow box of the plan, the user can select

- tick the boolean box that state whether the plan is accomplished or not;
- list the plans with which such a plan is in conflict (within the same unit, that can be minimal or not);
- list the plans which such a plan is in support of (within the same unit, that can be minimal or not);
- list the values that are put at stake by such a plan;
- list the values that are put in balance by such a plan.

5 POP-ODE conversion from the data base to the instantiated ontology: DB2OWL

5.1 Description

The DB2OWL package is coded in Java. The goal of this package is to convert from the current data base to the instantiated ontology (Drammar Instantiated Ontology, DIO file). The $DB2OWL$ folder is composed of the following material:



```
Act1, Scene1

Enter BARNARDO and FRANCISCO,
      two sentinels, [meeting].


   BARNARDO

1  Who's there?

   FRANCISCO
2  Nay, answer me: stand, and unfold yourself.
```

```
    POLONIUS
88  What is't, Ophelia, he hath said to you?

    OPHELIA
89  So please you, something touching the Lord Hamlet.

    POLONIUS
90  Marry, well bethought:
91  'Tis told me, he hath very oft of late
92  Given private time to you; and you yourself
93  Have of your audience been most free and bounteous:
94  If it be soas so 'tis put on me,
```

annotatore*: Dal Bello

Previous unit

name	Unit1_III_2_DAL
description	Laertes riceve la benedizione dal padre Polonius prima della partenza

value at stake before
no value
affetti familiari - Laertes
ambizione - Claudius
amicizia - Marcellus

value in balance before
no value
affetti familiari - Laertes
ambizione - Claudius
amicizia - Marcellus

Unit1_III_3_DAL

name	Unit1_III_3_DAL
description	Polonius chiede ad Ophelia dei suoi sentimenti

name*	type name
description	type description

Next unit

name	Unit1_III_4_DAL
description	Ophelia promette di non vedere più Hamlet

value at stake after
no value
affetti familiari - Laertes
ambizione - Claudius
amicizia - Marcellus

value in balance after
no value
affetti familiari - Laertes
ambizione - Claudius
amicizia - Marcellus

agent

name	Ophelia
description	type description

goal

name	uscire con Hamlet
description	

plan

convincere Polonius che Hamlet è affidabile

name	convincere Polonius che Hamlet è affidabile
description	
action	

relations

accomplished? yes [✓ no]

in conflict with
no plan
spiegare a Ophelia che è troppo inger

Fig. 2. The web interface of the POP-ODE annotation: top) text selection; middle) unit annotation; bottom) plans-agents-goals-conflicts-values annotation.

– folder *src*, containing the source files;
– folder *bin*, containing the class files;
– folder *lib*, containing the libraries used by the package;
– file *DrammarSQLManager.xml*, containing the parameters of the db connection, to be updated as for the previous section;
– a few owl files: some are read-only files; the package will update the file *drammar_ct.owl*, to be addressed later below;
– file *print_out.txt*, containing the result of the execution, with the possible errors (annotators can search for the string "Error" within the file and correct appropriately).

5.2 Usage

There are two command files (for Mac users; other platform users can write different batch files), one for compiling the code (*compileDrammarMgr.command*), one for executing the code (*startDrammarMgr.command*). The execution of the code produces two files, *drammar_ct.owl* and *print_out.txt*. Open the file *print_out.txt* with a text editor, to check for errors and correct them if they are substantial for your task. Then, open the file *drammar_ct.owl* with the Protègè editor and start reasoning (Pellet and Hermit have been tested); check and correct possible inconsistencies. Finally, export the inferred axioms (menu File) by selecting all the checkboxes except the one on Disjoint Classes (which seems to work badly in the current implementation) and set the local exported file to the name *drammar_ct_inferred.owl*. The last action is to copy the file *drammar_ct_inferred.owl* into the *OWL2CHART* folder.

Summing up, if no error occurs, for a Mac user, the steps are the following (while MAMP server is on):

1. execute file *startDrammarMgr.command*;
2. open the file *drammar_ct.owl* with the Protègè editor and reason;
3. export the inferred knowledge base to the file *drammar_ct_inferred.owl*;
4. copy the file *drammar_ct_inferred.owl* into the *OWL2CHART* folder.

6 POP-ODE conversion from the instantiated ontology to chart XML: OWL2CHART

6.1 Description

The OWL2CHART package is coded in Java. The goal of this package is to convert the instantiated ontology (DIO file, in OWL format) to the an XML file that represents the elements to be display as a characters' intentions chart (see next section). The *OWL2CHART* folder is composed of the following material:

– folder *src*, containing the source files;
– folder *bin*, containing the class files;
– folder *lib*, containing the libraries used by the package;
– two owl files: *drammar_ct_inferred.owl* is copied here after having been computed by the DB2OWL package; *drammar_ct_inferred_2.owl* is a by-product of the OWL2CHART package;
– XML file *drama_test.xml*, that contains the elements to be displayed (see next section);
– file *print_out.txt*, containing the result of the execution, with the possible errors (annotators can search for the string "Error" within the file and correct appropriately).

6.2 Usage

There are two command files (for Mac users; other platform users can write different batch files), one for compiling the code (*compileDrammarMgr.command*), one for executing the code (*startDrammarMgr.command*). The execution of the code produces three files, *drammar_ct_inferred_2.owl*, *drama_test.xml*, and *print_out.txt*. *drammar_ct_inferred_2.owl* is only a by-product of the conversion. The file *print_out.txt* (to be opened with a text editor) contains prints of the conversion for monitoring purposes; check for possible errors in the conversion and correct them if they are substantial for your task. The last action is to copy the file *drama_test.xml* into the *VisTool* folder, which contains the Processing sketch for visualization.

Summing up, if no error occurs, the steps are the following:

1. execute file *startDrammarMgr.command*;
2. copy the file *drama_test.xml* into the *VisTool/sketch_Drammar_vistool_13/data* folder.

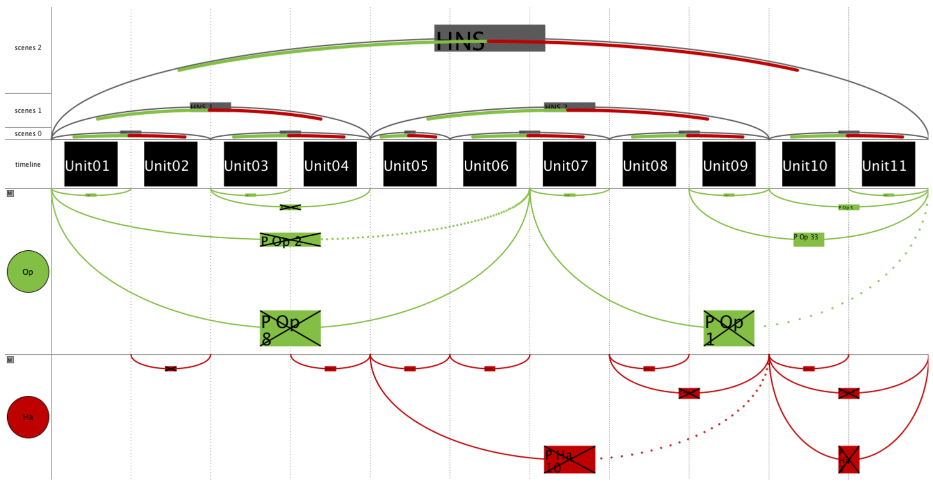


Fig. 3. Visualization of Hamlet nunnery scene. Through the tooltip facility of the interface we have magnified and highlighted the contents of the nodes (proportionally with their extension in the timeline).

.

7 POP-ODE visualization tool: VisTool

Figure 3 shows a detailed visualization of the "nunnery" scene (from Hamlet).

7.1 Description

The VisTool package is coded in Processing. The goal of this package is to display the characters' intentions aligned with units and scenes. XML file *drama_test.xml* contains the elements to be display as a characters' intentions chart. The $VisTool$ folder only contains the *sketch_Drammar_vistool_*13 folder. The reason for having this intermediate folder is because the Processing environments requires that the main file of the program must have the same of the name it is contained in. The *sketch_Drammar_vistool_*13 folder is composed of the following material:

– folder *data*, containing the files of the data to be visualized; in particular, though the folder contains a number of XML files, the only one that is currently visualized is *drama_test.xml*; so, if one wants to display some particular drama, it is enough to duplicate the corresponding file and name it *drama_test.xml*;
– a number of pde files (Processing Development Environment files): the main file is *sketch_Drammar_vistool_*13*.pde*, i.e. the one with the same name of the folder.

7.2 Usage

To launch the vistool, it is enough to click twice on the main file: this opens the Processing environment. One can have an interactive and a non-interactive display, respectively. The two executions are determined through the setting of the variable *interactive* to *true* or $false$, respectively. The non interactive execution produces a *png* file; the interactive execution opens an interactive window to be explored.

Summing up, if no error occurs, the steps are the following:

1. set the *interactive* variable within the *sketch_Drammar_vistool_13.pde* file to the desired execution;
2. click on the *play* symbol to produce the display.

References

1. Albert, G., Pizzo, A., Lombardo, V., Damiano, R., Terzulli, C.: Bringing authoritative models to computational drama (encoding knebel's action analysis). In: Interactive Storytelling. 9th International Conference on Interactive Digital Storytelling, ICIDS 2016. vol. 10045, pp. 285–297. Springer International Publishing, Cham – CHE (November 15–18 2016), http://dx.medra.org/10.1007/978-3-319-48279-825
2. Lombardo, V., Battaglino, C., Pizzo, A., Damiano, R., Lieto, A.: Coupling conceptual modeling and rules for the annotation of dramatic media. Semantic Web Journal 6(5), 503–534 (2015)
3. Lombardo, V., Damiano, R., Pizzo, A., Terzulli, C.: The intangible nature of drama documents: an FRBR view. In: Proceedings of the 2017 ACM Symposium on Document Engineering, DocEng 2017, Valletta, Malta, September 4-7, 2017. pp. 173–182 (2017), http://doi.acm.org/10.1145/3103010.3103019
4. Lombardo, V., Pizzo, A., Damiano, R.: Safeguarding and accessing drama as intangible cultural heritage. ACM Journal on Computing and Cultural Heritage 9(1), 1–26 (2016)
5. Lombardo, V., Pizzo, A., Damiano, R., Terzulli, C., Albert, G.: Interactive chart of story characters' intentions. In: Interactive Storytelling, 9th International Conference on Interactive Digital Storytelling, ICIDS 2016, Los Angeles, CA, USA, November 15–18, 2016, Proceedings. vol. 10045, pp. 415–418. Springer International Publishing, Cham – CHE (November 15–18, 2016), http://dx.medra.org/10.1007/978-3-319-48279-8_39