Web-STAR: Towards a Visual Web-Based IDE for a Story Comprehension System

Christos Rodosthenous and Loizos Michael

Open University of Cyprus
{christos.rodosthenous,loizos.michael}@ouc.ac.cy

Abstract. In this work, we present Web-STAR, an online platform for story understanding built on top of the STAR (STory comprehension through ARgumentation) reasoning engine. This platform includes a web-based IDE, integration with the STAR system and a web service infrastructure to support integration with other systems that rely on story understanding functionality to complete their tasks. The platform also delivers a number of "social" features like public story sharing with a built-in commenting system, a public repository for sharing stories with the community and collaboration tools that can be used from both project team members for development and educators for teaching. Moreover, we discuss the ongoing work on adding new features and functionality to this platform.

Keywords: web-based IDE, story understanding, argumentation, reasoning, visual programming, collaboration

1   Introduction

It is important for a logic-based system to have an intuitive interface that allows both expert and non-expert users to use it for preparing programs, debugging them and learning to use the provided features. In this work, we focus on such a system, the STAR (STory comprehension through ARgumentation) reasoning engine [12,11] that deals with the problem of story comprehension and we present a platform built on top of it.

Understanding stories, is one of the many tasks humans can perform well. Machines are still not capable of performing such a task as well as humans do, since it requires the ability to infer about states and events that are not explicitly described in the text [22]. This process is called commonsense reasoning and it requires knowledge about the world we live in (background knowledge).

Classical logic and variance has been used to represent commonsense knowledge and to give semantics for the reasoning process. Recent work [16] has suggested the use of argumentation [3,4] as a more appropriate framework for the representation and reasoning with commonsense knowledge.

In this work, we focus on the design and development of the **Web-STAR** platform. This platform includes a web-based integrated development environment (IDE) that presents a personalized environment for each user with tools

for writing, executing, debugging stories using the STAR system and visualizing their output. Moreover, the IDE is the basis of a community, where people can share stories, comment on them and reuse other community created stories. Under the same umbrella, a web service is also available for integrating other systems with the Web-STAR platform.

For increasing the usage and friendliness of the Web-STAR IDE, a number of design options are considered. These design options allow novice users to prepare their stories and add background knowledge in a more graphical or visual way, e.g., by using a directed graph representation of background knowledge and by using wizards to add questions and story related information.

In the following sections, we present the current state in story understanding systems and web IDEs followed by a presentation of the STAR system that is the underlying engine of the Web-STAR platform. Next, the Web-STAR platform is presented using a realistic scenario of using its IDE. In the final section, new features and additions to the Web-STAR platform are discussed as part of the ongoing work on the platform.

2   Related Work

In this section, we present related work on both story understanding systems and web-based IDEs, accompanied with examples of both general language and logic-based IDEs.

2.1   Story Understanding Systems

Currently, several systems have been developed to deal with the problem of story comprehension. Starting from the 70's, Charniak presented two systems; a story comprehension model for answering questions about children stories, by relating stories to real-world background knowledge [5] and a system for answering questions about simple stories dealing with painting [6]. There is also work on Deep Read [14], an automated reading comprehension system that accepts stories and answers questions about them. More recent attempts, include work of Mueller [23], on a system for modeling space and time in narratives about restaurants and Genesis [30], a system for modeling and exploring aspects of story understanding using stories drawn from sources ranging from fairy tales to Shakespeare's plays.

In terms of technical skills and expertise needed to use these systems, the majority of them use a command line interface (CLI), except Genesis that already has a visual interface. The lack of a visual environment makes it difficult for non-experts to setup and use these systems without prior programming knowledge and explicit knowledge of the specific system. Furthermore, the majority of these systems rely on external tools (text editors) for editing the source code and lack basic functionality that an IDE can easily provide (i.e., code folding, syntax highlighting, etc.).

2.2 Web-Based IDEs

Developments in web technologies gave birth to numerous web-based IDEs, some of which are now considered as mainstream IDEs for developing applications. These are systems that are available through a web browser, they do not rely on specific hardware or software stack and are Operating System agnostic, e.g., Cloud9 [7], Codiad [25], ICEcoder [15], codeanywhere [8] and Eclipse Che[13].

These web-based IDEs allow users to write code in an online source code editor using their programming language of choice. They also provide code folding, code highlighting and auto-complete functionality, built-in their source code editors. Moreover, some of them provide online code execution functionality with access to an underlying virtual machine and thus access to the shell.

There are also IDEs concentrated on logic programming, like SWISH (SWI-Prolog for Sharing) [28] and IDP Web-IDE [10]. SWISH is a web front-end for SWI-Prolog which is used to run small Prolog programs for demonstration, experimentation and education. IDP Web-IDE is an online front-end for IDP, a Knowledge Base System for the FO($\cdot$) language.

3   The STAR System

Before presenting the Web-STAR IDE, it is important to have an overview of the STAR system, the underlying engine used for story comprehension, and its semantics. The STAR system, is based on the well established argumentation theory in Artificial Intelligence [3,2], uniformly applied to reason about actions and change in the presence of default background knowledge [12]. The STAR system follows guidelines from the psychology of comprehension, both for its representation language and for its computational mechanisms for building and revising a comprehension model as the story unfolds.

The system takes as input a story written in symbolic predicate form using time-stamped facts, background knowledge in the form of association rules (e.g., `bird(X) implies fly(X)`) accompanied by priorities between them, and questions on the story facts. Each question can be asked as part of several distinct sessions, to check if there has been a revision in the comprehension model of the system as the story progresses. The output of the system includes the comprehension model and answers to the multiple choice questions posed for each session.

The STAR system is a Prolog application that runs in any environment which supports SWI-prolog [29]. It requires users to setup the Prolog environment, call the application's core engine file and select a domain file to load. Users can also choose what information is reported by the system during its execution.

A domain file follows the STAR system syntax and comprises two main parts; The "story and questions part" that includes a series of sessions for representing the story narrative up to a scene (session), a set of questions up to that point, and the "background knowledge part" that includes knowledge rules about the context of the story.

In the following paragraphs, we present the semantics of the STAR system using the following story in natural language:

> Ann rang the doorbell. Mary, who was in the flat watching TV, got up from her chair and walked to the door. She was afraid. Mary looked through the keyhole. Mary saw Ann, her flatmate, at the door. Mary opened the door and asked Ann why she did not use her keys to come in the flat? Ann replied that she is upset that she did not agree to come with her at the shops. She wanted to get her up from her chair in front of the TV.

The first part of the file encodes the various observations captured in the story with time-points for each one of them. The story content of a session is given by a set of observation statements of the form `s(#N) :: #GroundLiteral at #Time-Point`, where `#N` is a non-negative integer representing the session. A literal `#Literal` is either a concept `#Concept` or its negation `-#Concept` (i.e., the symbol for negation is "-"), where a concept `#Concept` is a predicate name along with associated variables or constants for the predicate's arguments.

For example, the above short story could be represented as:

```
s(0) :: person(ann) at always.
s(0) :: person(mary) at always.
s(0) :: request(go_to(shops)) at always.
s(0) :: request(donate(money)) at always.
s(0) :: ring(ann,doorbell) at 2.
s(0) :: in_flat(mary) at 3.
s(0) :: watch(mary,tv) at 3.
...
```

The story plot is presented in a single session, thus the `s(0)` notation in front of every observation statement. The first observations state that "ann" and "mary" are instances of type "person" and that holds at every time-point. After the observation statements, the questions are posed. A question in the STAR system follows this notation: `q(XX) ?? PREDICATE1 at TIME-POINT; PREDICATE2 at TIME-POINT.`, where XX is a non-negative integer representing the number of the question and the ";" separates the possible answers to that question. For the above story two example questions could be:

```
% Question 1: Does Ann have the door keys?
q(1) ?? has(ann,doorkeys) at 1. % True or False type question
```

```
% Question 2: Why did Mary walk to the door?
% Possible answers: Because she wants to find out who is at the door or
because she wants to open the door
```

```
q(2) ?? wants(mary,find_out_who_at(door)) at 4;
        wants(mary,open(door)) at 4.
```

The second part of the domain file, includes the background knowledge required to comprehend the story. Background knowledge is represented in terms of associations between concepts. The STAR system supports three types of rules, property, causal, preclusion, of the following form respectively:

```
p(#N) :: #Body implies #Literal.
c(#N) :: #Body causes #Literal.
r(#N) :: #Body precludes #Literal.
```

The body of the rule is in the form of `#Body = true | #Literal | #Literal,` `#Body`. Background knowledge can also contain priority (i.e., relative strength) statements between the rules. Priorities are encoded in the form #P1 >> #P2., where #P1 and #P2 are the names of two rules. The following is an example of such background knowledge used in the above story:

% Walking to the door implies wanting to open the door, unless one is afraid.
```
p(22) :: walk_to(Person,door), in_flat(Person) implies
         wants(Person,open(door)).
p(23) :: afraid(Person), in_flat(Person) implies
         -wants(Person,open(door)).
```
p(23) >> p(22). % this captures the precedence of p(23) over p(22)
% Seeing a flatmate at the door causes one to want to open the door.
```
c(33) :: see_at(Person,Other,door), flatmate(Person,Other) causes
         wants(Person,open(door)).
```
c(33) >> p(23). % even if one is afraid.
...

Before completing the story writing process, users need to define which questions will be answered in each session and which domain concepts should be visible on the screen as the system constructs its comprehension model. This is done using a session statement at the beginning of the story file:

% session(s(#N),#Questions,#Visible).
```
session(s(0),[q(1),q(2)],all).
```

Users also need to declare the concepts that are fluents, i.e., persist over time and change across time. This is done using a statement of the form: `fluents(#Fluents)`, where #Fluents is a list of fluents.

More details about the STAR system can be found in relevant work of Diakidou et al. [12] and on the system's website. The full example used in this section, is available at the "Help" menu of the Web-STAR IDE. Readers interested in the technical aspects of the argumentation semantics of the STAR System are directed to work of Diakidou et al. [11].

4   A Web Interface For The STAR System (Web-STAR)

The Web-STAR IDE is available online (http://cognition.ouc.ac.cy/star/) and is accessible from any device. Users are presented with a web interface that includes the source code editor, textual and visual output of the comprehension model, a number of control and debugging options and online help.

Moreover, the IDE employs a number of social features for user collaboration, like public code sharing and posting of stories to the public repository. In addition, users can work together using a state-of-the-art collaboration utility that allows screen sharing, text and voice chat and presenter following functionality. In short, work on Web-STAR includes:

– A WEB-IDE that does not require setup, is OS agnostic and offers most modern IDEs' functionality
– A platform for collaboration and teaching
– A platform for integrating story comprehension functionality to other systems

Figure 1, depicts the architectural diagram of the Web-STAR platform that comprises the Web-STAR IDE, the web services, the STAR system engine, the public repository and the databases for storing related information. In the next paragraphs, an example of using the web-based IDE is depicted using a scenario where "Nick", a university student, wants to learn how to use the STAR system for running experiments on story comprehension.
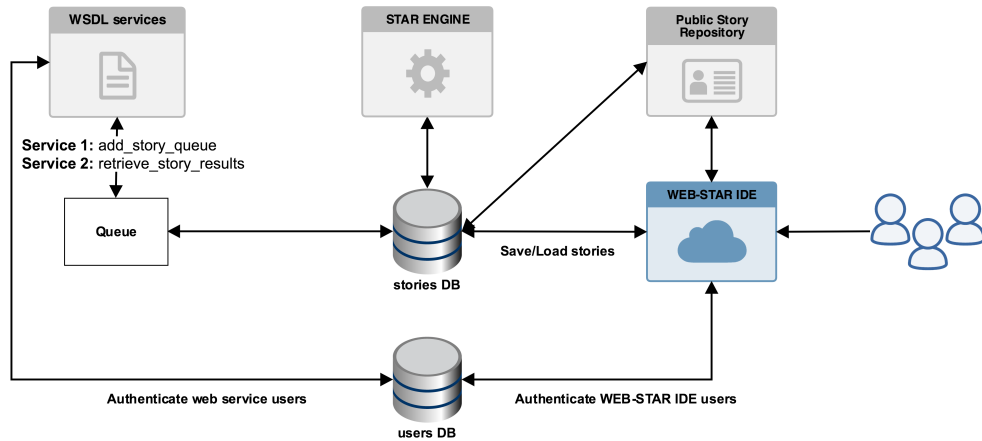

Fig. 1: The Web-STAR system architecture. The diagram includes the authentication mechanism, the storage functionality and the web services provided.

The IDE Environment and Workspace First, he creates an account to use the system and activates the personal workspace. Currently, both local and remote authentication options are available. The local authentication method uses the integrated storage facilities of the platform and remote authentication uses the OAuth2 protocol (https://oauth.net/2/), offered by third parties like Facebook, Google, Github, etc.

After the authentication process is complete, he is redirected to the source code editor, where he can write or load code and edit it. The source code editor is based on ACE editor (https://ace.c9.io), an open source web editor used also by many other popular cloud IDEs. This editor was chosen because of its maturity and open source license. Ace is a code editor written in JavaScript and includes features like syntax highlighting, theming, automatic indent and outdent of code, search and replace with regular expressions, tab editing, drag-drop functionality, line wrapping and code folding. Moreover, this editor can handle huge documents with more than one million lines of code.
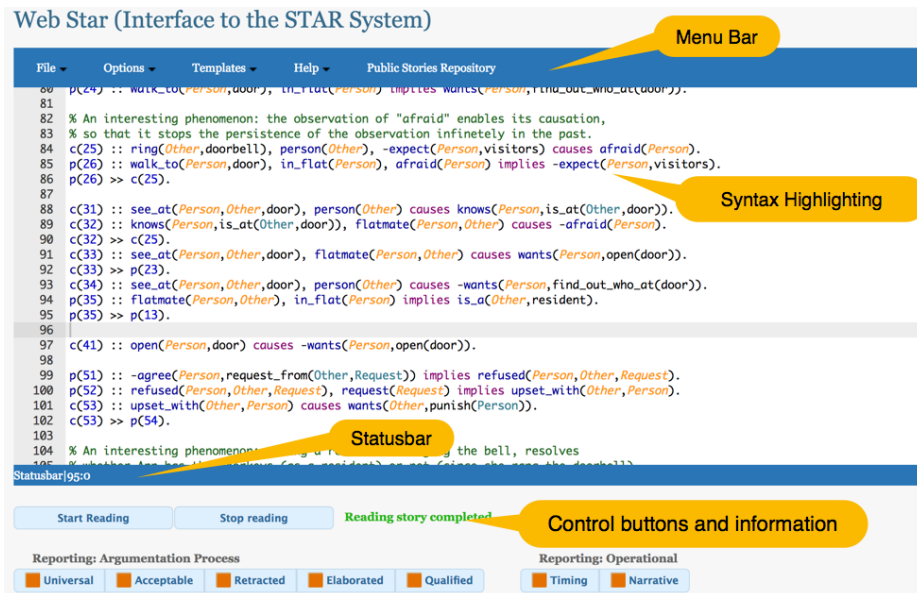

Fig. 2: A screenshot of the source code editor, depicting the line numbering, syntax highlighting and line highlighting. Above the source code editor resides the menu and at the bottom is the statusbar.

More specifically, users can write their code from scratch in the source code editor, load it from an external file or load an example file. Non-experts like Nick, can benefit from the many examples available. Currently, the source code editor allows syntax highlighting using a STAR syntax highlighter file that inherits the

Prolog's syntax template. Furthermore, line numbering and code wrapping are also available to users.

The Web-STAR IDE has a comprehensive list of menu options that enable users to load example story files, study them and edit them. Users have a personal workspace for saving their newly created programs and a public workspace for loading other user's programs. A program that is saved in the personal workspace can only be accessed by its creator, whereas a program stored in the public space, is visible to everyone. Options for importing code stored locally on the user's personal device and exporting stories to a file for local processing are also available. This functionality allows a user to use a locally installed version of the STAR system to process the story.

Nick, loads the example story presented in Section 3. The source code editor immediately identifies and highlights the STAR semantics (variables, rules, operators, etc.) and makes it easy for Nick to understand the example (see Figure 2). After studying the file, he moves to the questions part and he decides to try and add one more question by choosing the "question template" from the menu. The question template (presented in the previous Section) is added and Nick adds the predicates and time-points where the question is posed. Since he has done changes to the example file, he now saves it to the personal workspace using the corresponding menu option.

Story Comprehension and Output Nick completes the changes in the story and he tries to process it. He clicks the "start reading" button and he immediately sees results coming from the STAR system in the "Raw output" tab. He chooses the Graphical output since its easier for a non-expert to follow the changes as the story progresses through the different time-points.

Results include both the comprehension model and answers to the questions posed in the story file. Web-STAR presents its output both in textual and graphical format. The output area comprises two tabs, one for the raw output and one for the visual output. Each tab is dynamically updated when new information is sent from the STAR system.

In Figure 3, the graphical output of the comprehension model is depicted, presenting the state of each concept at each time-point. Green represents concepts whose value is positive at that time-point, red represents concepts whose value is negative at that time-point and dark grey represents concepts whose value is unknown at that time-point. The magnifying glass is for concepts whose value is observed at that time-point, i.e., they are extracted from the narrative content and are not inferred. Concepts with orange background represent an instantaneous action, concepts with light blue background represent a persisting fluent and concepts with purple background represent a constant type (e.g., `person(ann) at always.`).

Reporting Options Further studying on how the STAR system works for story comprehension, made Nick curious on the internal processes and more specifically on how the argumentation mechanism is applied for story comprehension.
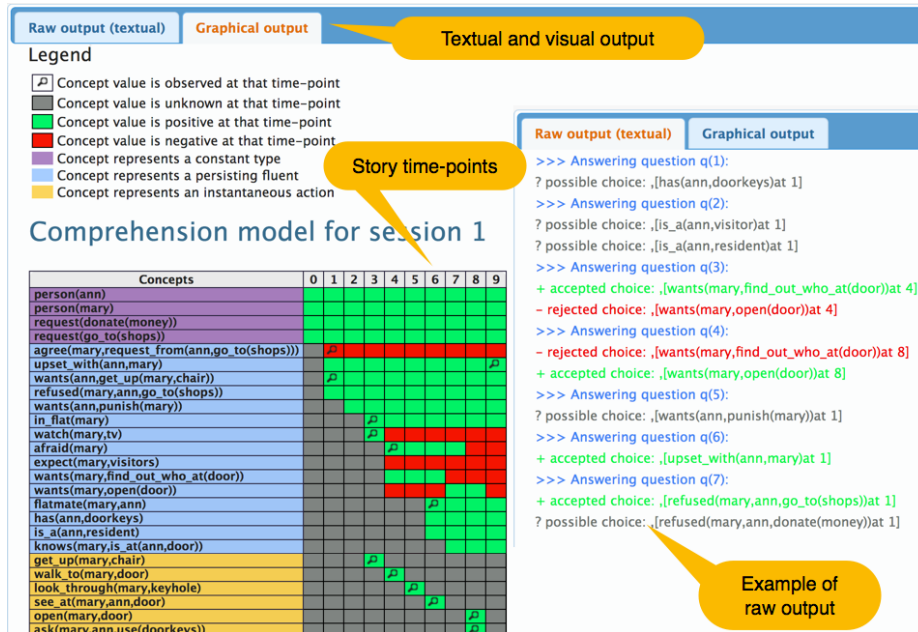
Fig. 3: A screenshot of the "Graphical output" tab of the IDE, showing the comprehension model. The legend above the comprehension model provides details on the meaning of symbols and colors in the graphical model. On the right side, the raw output for the same story is presented.

He decides to enable the reporting options. Users have the option to enable or disable reporting functionality used for debugging. In particular, users can choose to view all arguments (`Universal`), a subset of the acceptable arguments (`Acceptable`), arguments that are removed from a specific session (`Retracted`), arguments added in a specific session (`Elaborated`) and information about which arguments qualify other arguments (`Qualified`).

Collaboration and "Social" Options Apart form the "expected" IDE functionality, Web-STAR also provides functionality for sharing publicly a story file for others to use as an example. Users can share a story by clicking the "Share it" button. This story becomes visible to the public stories repository (see Figure 4) and the public stories menu. Users can add comments on a story and read it using the built-in functionality. This is used primarily for educational purposes, from users that are trying to learn and would benefit from the opinions of other more expert users.

Moreover, users can collaboratively write a story using the collaboration functionality provided. The system produces a link that can be send to anyone interested in collaborating for a specific session through an email or an instant

messaging application. Anyone with the link can see the screen, mouse pointers of each participating user, changes of content in real-time and can chat using both text and audio. This setting enables teams to collaborate for creating a story file and students for learning and working together on class projects.

Nick thinks that the story he created can be useful to other students and he shares it publicly by clicking on the "Share it" button. Not long after, some of his classmates added some comments and a discussion around the story starts.
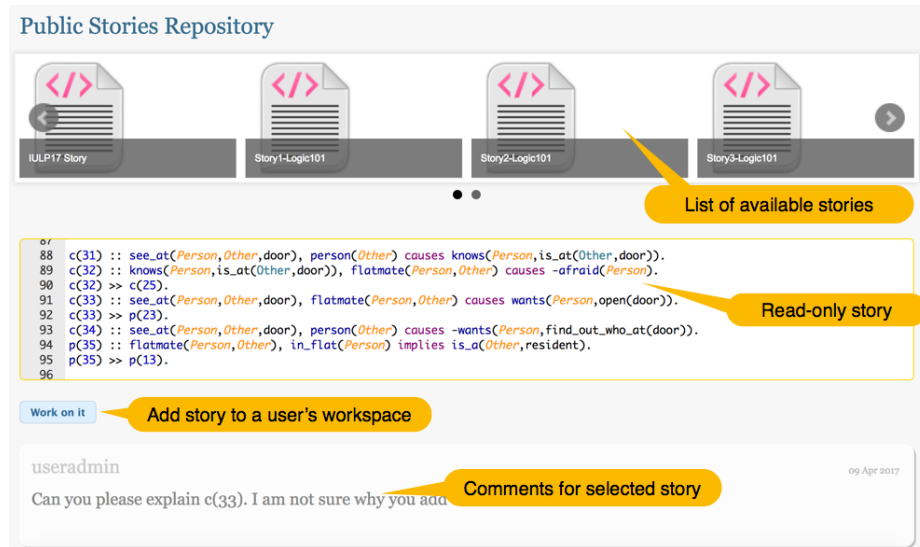

Fig. 4: A screenshot of the Public Stories Repository. Users can add comments on stories and ask questions. If interested, they can start working on a story by copying it to their personal workspace.

4.1   System Implementation

The system is based on PHP for backend operations, MariaDB database for data storage and JQuery javascript library for the front-end design. Behind this infrastructure lies the STAR system [12] and the Prolog interpreter. A wrapper is employed for sending the story file from the front-end to the back-end and returning the results in real-time from the Prolog interpreter using the HTML5 "Server-Sent Events" functionality to dynamically update the interface.

All data storage is handled through the MariaDB database. In particular, a number of tables are used for storing user data, user profiles and the STAR web service queue.

For the front-end design, HTML, JavaScript and CSS3 are used to present an easy to use web environment. The JQuery library (https://jquery.com/) is

also used to add intuitive UI components and AJAX functionality. Collaboration functionality is provided using both AJAX components for sharing and commenting on stories and the TogetherJs library (https://togetherjs.com/).

Web Services The Web-STAR platform exposes two web services that can be used by third party applications for adding a domain file to the STAR system queue for processing and retrieving the results of the reasoning process when completed (see Figure 1). This approach was chosen to minimize the waiting time in cases of large story files that require extensive processing.

4.2   Platform Usage Metrics and Examples

The Web-STAR platform is available online. Currently, more than thirty users have registered and contributed more than 50 stories. Moreover, the system has received more than 1350 web service calls for processing STAR domain files.

The platform was also used in the Robot Trainer Game [24] for real time processing of the acquired knowledge rules. Robot Trainer is a Game With A Purpose (GWAP) that uses a hybrid approach for acquiring knowledge in the form of rules using both human contributors and the Web-STAR platform. Moreover, these rules were evaluated by humans for their applicability on answering questions about stories different than the ones used to generate them.

5   Ongoing Work

Moving towards the direction of maximizing the number of people that are able to use a story comprehension system, we are currently working on a number of visualization additions to the Web-STAR IDE, which could eventually be used in other domains as well. Some of these additions are already developed to match the needs of other systems and are currently being integrated to the Web-STAR IDE. In the following paragraphs, these additions are presented with the relevant discussion on the advantages of using them. Currently, we are still investigating the various approaches that could be used in this direction.

5.1   Visualizing the Background Knowledge

Some of the processes that could be visualized include the previewing, addition and editing of rules. For these processes a directed graph with nodes and edges could be used. For example, the rules presented in the knowledge base snippet below can be represented by the directed graph in Figure 5:

```
r01 :: a, -z implies c.
r02 :: a implies -c.
r03 :: -c implies p.
...
r01 >> r02.
r19 >> r10.
```
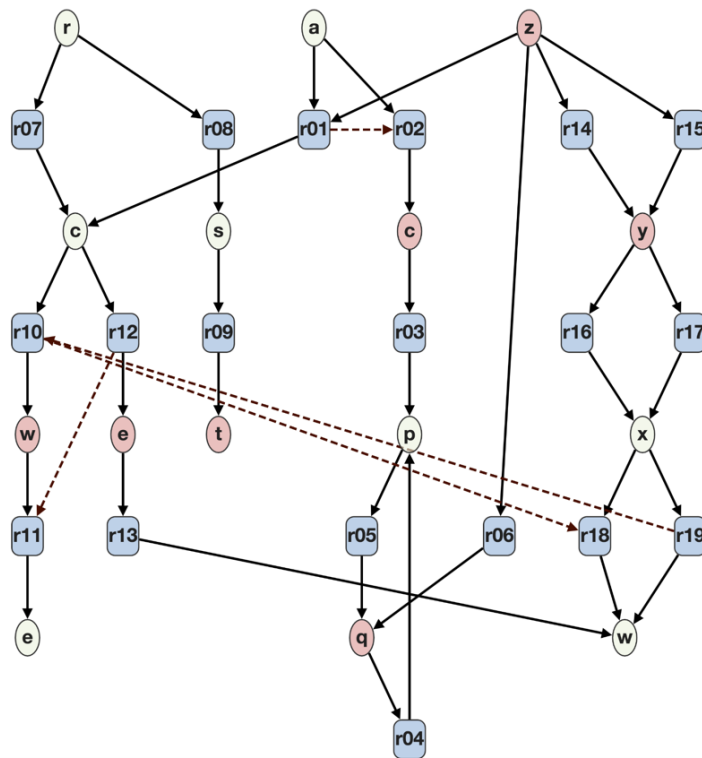
Fig. 5: An indicative example of a directed graph that can be used for representing visually the background knowledge. This screenshot is from a prototype version of the Web-STAR IDE that is under development and will soon be released. More specifically, the positive and negative concepts are presented using green and red nodes respectively and preferences between rules are presented with a dashed edge between nodes.

Users are able to change a rule by just clicking on the corresponding edge or node in order to remove or edit it. Users also have the ability to observe rules that build on each other i.e., rules that the predicates in the head exist in another rule's body, and better understand the reasoning process. For adding a rule, users will still have to type a compatible with the STAR system semantics name and add predicates for the head and body of the rule.

Since some of these story files might grow big, it would be useful for the user to isolate the part of the background knowledge that wants to work with at a given time. To support this, a type of code folding functionality needs to be implemented, which will allow users to focus on a specific group of rules in the screen and "minimize" the others, either by hiding the head and body of each rule or by showing them in a smaller size.

This functionality can also be used for feedback on the reasoning process. For example, a user can choose to change the priority between two rules by just clicking on the edge that connects them to remove it and add a new edge between different rules. Such functionality could be useful in other domains, where users need to give their feedback and change the knowledge rules and priorities stored in the system.

In particular, we are examining the addition of this functionality in a Game platform, where users will train their "Player" on what strategy to choose for winning the game. This platform will enable game players to visually add rules and priorities without any prior knowledge of the underlying reasoning system.

5.2   Support for Natural Language Input

Aiming in a fully automated solution, a number of problems must be addressed before deploying such a system. These problems involve the encoding of natural language text to symbolic form suitable for the STAR system, event extraction from text, temporal event ordering and acquisition of background knowledge suitable for a selected story.

A number of these problems, can be addressed by incorporating well established systems to the IDE's toolset. Systems like PIKE [9] that is able to extract knowledge from natural language into an RDF triplet using Semantic Role Labeling [27] and Stanford CoreNLP [19] that can process a text in natural language and extract knowledge information, are good candidates for this task. Furthermore, there are a number of techniques that can be used for temporal event ordering [27]. More difficult tasks, like background knowledge acquisition, can be sourced to the crowd, using Games With A purpose [1] or other crowd based techniques, or can be addressed through machine learning approaches [20]. Even Knowledge Bases like Conceptnet [26], YAGO [18] and NELL [21] can be used for this task or even manually coded rules from the OpenCyc project [17].

6   Conclusions

In this work, we described the Web-STAR platform, built on top of the STAR system for story comprehension. Information on the technologies used for implementing the platform, the user interface and the collaboration tools available are presented. Moreover, a number of examples were used to explain how to use the Web-STAR IDE to prepare a story and a use case scenario was used to elaborate on how to process a story. The platform is currently used by University students and Instructors for educational purposes and as part of a GWAP, which benefits from the platform's web services for answering questions.

Ongoing work on this platform aims in enabling non-expert users to easily use and experiment with the STAR system. These additions will also provide the basis of the feedback module needed for integrating the STAR or other similar engine with other systems.

References

1. von Ahn, L., Dabbish, L.: Designing games with a purpose. Communications of the ACM 51(8), 57 (2008)
2. Baroni, P., Caminada, M., Giacomin, M.: An Introduction to Argumentation Semantics. Knowl. Eng. Rev. 26(4), 365–410 (2011)
3. Bench-Capon, T., Dunne, P.E.: Argumentation in artificial intelligence. Artificial Intelligence 171(10-15), 619–641 (2007)
4. Besnard, P., Hunter, A.: Elements of argumentation, vol. 47. MIT press Cambridge (2008)
5. Charniak, E.: Toward a Model of Children's Story Comprehension. Tech. Rep. AITR-266, Cambridge, MA, USA (1972)
6. Charniak, E.: Ms. Maloprop, a Language Comprehension Program. In: Proceedings of the 5th International Joint Conference on Artificial Intelligence - Volume 1. pp. 1–7. IJCAI'77, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1977)
7. Cloud9 IDE, L.: Cloud9 IDE (2017), `https://c9.io`
8. Codeanywhere Inc.: codeanywhere IDE (2017), `https://codeanywhere.com/`
9. Corcoglioniti, F., Rospocher, M., Palmero Aprosio, A.: Frame-Based Ontology Population with PIKES. IEEE Transactions on Knowledge and Data Engineering 28(12), 3261–3275 (2016)
10. Dasseville, I., Janssens, G.: A web-based IDE for IDP. In: International Workshop on User-Oriented Logic Programming (IULP) (2015)
11. Diakidoy, I.A., Kakas, A., Michael, L., Miller, R.: Story Comprehension through Argumentation. In: Proceedings of the 5th International Conference on Computational Models of Argument (COMMA 2014). pp. 31–42. Scottish Highlands, UK (2014)
12. Diakidoy, I.A., Kakas, A., Michael, L., Miller, R.: STAR: A System of Argumentation for Story Comprehension and Beyond. In: Working notes of the 12th International Symposium on Logical Formalizations of Commonsense Reasoning (Commonsense 2015). pp. 64–70 (2015)
13. Eclipse Foundation: Eclipse Che IDE (2017), `https://eclipse.org/che/`
14. Hirschman, L., Light, M., Breck, E., Burger, J.D.: Deep Read: A reading comprehension system. In: Proceedings of the 37th annual meeting of the Association for Computational Linguistics on Computational Linguistics. pp. 325–332. Stroudsburg, PA, USA (1999)
15. ICEcoder Ltd: ICEcoder IDE (2017), `https://icecoder.net/`
16. Kakas, A., Michael, L.: Cognitive Systems: Argument and Cognition. IEEE Intelligent Informatics Bulletin 17(1), 14–20 (2016)
17. Lenat, D.: CYC: A Large-Scale Investment in Knowledge Infrastructure. Communications of the ACM 38(11), 33–38 (1995)
18. Mahdisoltani, F., Biega, J., Suchanek, F.M.: YAGO3: A Knowledge Base from Multilingual Wikipedias. Proceedings of CIDR pp. 1–11 (2015)
19. Manning, C.D., Bauer, J., Finkel, J., Bethard, S.J., Surdeanu, M., McClosky, D.: The Stanford CoreNLP Natural Language Processing Toolkit. In: Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations. pp. 55–60 (2014)
20. Michael, L.: Reading Between the Lines. In: Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI 2009). pp. 1525–1530 (2009)
21. Mitchell, T., Cohen, W., Hruschka, E., Talukdar, P., Betteridge, J., Carlson, A., Mishra, B.D., Gardner, M., Kisiel, B., Krishnamurthy, J., Lao, N., Mazaitis, K.,

Mohamed, T., Nakashole, N., Platanios, E., Ritter, A., Samadi, M., Settles, B., Wang, R., Wijaya, D., Gupta, A., Chen, X., Saparov, A., Greaves, M., Welling, J.: Never-Ending Learning. In: AAAI Conference on Artificial Intelligence. pp. 2302–2310 (2015)

22. Mueller, E.T.: Story Understanding Through Multi-representation Model Construction. In: Proceedings of the HLT-NAACL 2003 Workshop on Text Meaning - Volume 9. pp. 46–53. HLT-NAACL-TEXTMEANING '03, Association for Computational Linguistics, Stroudsburg, PA, USA (2003)

23. Mueller, E.T.: Modelling space and time in narratives about restaurants. Literary and Linguistic Computing 22(1), 67–84 (2007)

24. Rodosthenous, C., Michael, L.: A Hybrid Approach to Commonsense Knowledge Acquisition. In: Proceedings of the 8th European Starting AI Researcher Symposium. pp. 111–22 (2016)

25. Safranski, K.: Codiad Web Based, Cloud IDE (2017), http://codiad.com/

26. Speer, R., Havasi, C.: ConceptNet 5: A Large Semantic Network for Relational Knowledge, pp. 161–176. Springer Berlin Heidelberg, Berlin, Heidelberg (2013)

27. UzZaman, N., Llorens, H., Derczynski, L., Verhagen, M., Allen, J., Pustejovsky, J.: SemEval-2013 Task 1: TempEval-3: Evaluating Time Expressions, Events, and Temporal Relations. Second joint conference on lexical and computational semantics (* SEM) 2(SemEval), 1–9 (2013)

28. Wielemaker, J., Lager, T., Riguzzi, F.: SWISH: SWI-Prolog for Sharing. In: International Workshop on User-Oriented Logic Programming (IULP) (2015)

29. Wielemaker, J., Schrijvers, T., Triska, M., Lager, T.: SWI-Prolog. Theory and Practice of Logic Programming 12(1-2), 67–96 (2012)

30. Winston, P.H.: Model-based Story Summary. In: Finlayson, M.A., Miller, B., Lieto, A., Ronfard, R. (eds.) 6th Workshop on Computational Models of Narrative (CMN 2015). OpenAccess Series in Informatics (OASIcs), vol. 45, pp. 157–165. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany (2015)