TLP **19** (2): 317–359, 2019. © Cambridge University Press 2018
doi:10.1017/S1471068418000443 First published online 14 November 2018

*Web-STAR: A Visual Web-based IDE for a Story Comprehension System**

CHRISTOS T. RODOSTHENOUS
Open University of Cyprus, Nicosia, Cyprus
(*e-mail:* christos.rodosthenous@ouc.ac.cy)

LOIZOS MICHAEL
Open University of Cyprus and Research Center on Interactive Media, Smart Systems, and Emerging Technologies, Nicosia, Cyprus
(*e-mail:* loizos@ouc.ac.cy)

submitted 29 September 2017; revised 8 August 2018; accepted 15 August 2018

Abstract

We present Web-STAR, an online platform for story understanding built on top of the STAR reasoning engine for STory comprehension through ARgumentation. The platform includes a web-based integrated development environment, 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, including a community repository for public story sharing with a built-in commenting system, and tools for collaborative story editing that can be used for team development projects and for educational purposes.

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

1 Introduction

Due to its long history, automated story understanding (Mueller 2006), and text comprehension in general, has attracted interest from researchers across a diverse set of fields, including computer science, artificial intelligence, logic programming, psychology, language learning, narratology, and law. Researchers from these areas have varying interests in the different parts of the comprehension process and varying skills in interacting with an automated comprehension system that seeks to implement this process. For story comprehension systems that rely on a symbolic representation of knowledge, for instance, one could roughly distinguish two groups of users: expert users (e.g. computer

* The authors would like to thank: Adamos Koumis, for his collaboration in the development of the component that converts natural language to the STAR syntax; Elektra Kypridemou, for her help in the preparation of the evaluation methodology; and the anonymous reviewers, for their valuable comments and suggestions.
An earlier version of this work was presented at the 2nd International Workshop on User-Oriented Logic Paradigms (IULP 2017). This article presents a newer version of the Web-STAR IDE with additional implemented features, along with the results of a user evaluation conducted to verify the usability and learnability of the IDE.

https://doi.org/10.1017/S1471068418000443 Published online by Cambridge University Press

scientists, logic programmers, and AI experts), who might be more interested in developing systems for story understanding, are able to encode and read stories and background knowledge in a machine-readable format; non-expert users (e.g. psychologists, language experts, and narrators), who might be primarily interested in utilizing existing systems for story understanding, may prefer to write stories in natural language, to examine the comprehension process and perform experiments, without caring that much about the internal encodings and representations that are used by the automated systems.

The distinction that we make between experts and non-experts is not meant to be absolute. Junior computer science students might fit better in the non-expert category, and language experts might be considered experts for the particular task of translating a story into a logic-based representation, even if they lack the skills to handle other parts of a story comprehension system. In any case, the diversity and heterogeneity that exists in terms of expertise in the use of automated story comprehension systems suggests the need for systems with a simple and intuitive interface that allows expert and non-expert users to input and reason with chosen stories, and to trace and debug the comprehension process. In this work, we start from an existing story comprehension system and build a platform on top of it to endow the end system with the aforementioned characteristics.

We consider, in particular, an existing automated story comprehension system called STAR: STory comprehension through ARgumentation (Diakidoy *et al.* 2014; Diakidoy *et al.* 2015), which adopts the view that comprehension requires the drawing of inferences about states and events that are not explicitly described in the story text (Mueller 2003) through the use of background world knowledge and commonsense reasoning (Mueller 2015). Retaining the view that stories and background knowledge are symbolically represented, the STAR system abandons classical logic as the underlying semantics for knowledge and adopts argumentation (Bench-Capon and Dunne 2007; Besnard and Hunter 2008) as a more appropriate substrate for the development of automated systems that interact with humans (Kakas and Michael 2016; Michael 2017).

In this work, we present the design and development of the **Web-STAR** platform built on top of the STAR system. The platform includes a web-based integrated development environment (IDE) that presents a personalized environment for each user with tools for writing, comprehending, and debugging stories while visualizing the output of the comprehension process. Moreover, the IDE offers the basis for building a community, where people can share stories, comment, and reuse other community-created stories. Under the same umbrella, a web service is also made available for integrating other systems with the Web-STAR platform.

Web-STAR allows both expert and non-expert users to write stories and encode them in the internal STAR syntax, offering a number of features. Non-expert users can take advantage of the following features: (i) the automatic conversion of a story from natural language to the STAR syntax; (ii) the encoding of background knowledge using a visual representation based on directed graphs; (iii) the automatic conversion of the graph to the STAR syntax and vice versa. Non-expert users also benefit from the visual representation of the system output in a time-line format. Expert users benefit from the feature-rich IDE, which allows the preparation of a story in the STAR syntax using a state-of-the-art source code editor, the reasoner debugging options, and the raw output. All users benefit from the collaboration options available and the story repository.

https://doi.org/10.1017/S1471068418000443 Published online by Cambridge University Press

In the following sections, we present the current state of affairs on story understanding systems and web-based IDEs that are used in logic-based systems, followed by a presentation of the STAR system as the underlying engine of the Web-STAR platform. Next, the Web-STAR platform is presented with details of the various features that it offers, along with scenarios on how these features can be used. The platform's usability is then evaluated and discussed, and new features and additions to the Web-STAR platform are presented as part of our ongoing work on the platform.

2 Related work

In this section, we provide insights on the concept of story understanding and its relation to the notion of explanation and we present related work on story understanding systems and on web-based IDEs that are geared toward imperative and declarative (logic-based) languages. Currently, little work has been done to enhance story understanding systems with functionality present in an IDE, and more specifically in a web-based IDE.

2.1 Story understanding

There are many definitions of what a story or a narrative is (Genette 1982; Prince 2003; Abbott 2008; Ricoeur 1980; Ryan 2007), but in this context we use the view of Rick Altman (2008), as reported by Michael (2013), who argues that "virtually any situation can be invested with [those] characteristics [necessary to] perform the narrational function." Story understanding includes the human ability to answer arbitrary questions, generate paraphrases and summaries, fill arbitrary templates, make inferences, reason about the story, hypothesize alternative versions of the story, look back over the story, and more (Mueller 2000).

Story understanding requires many different capabilities, including language understanding, word sense disambiguation, information extraction, along with the existence of vast amounts of background knowledge (Davis 1990; Lenat and Guha 1990). In the work of Mueller (2000), a historical overview of the story understanding area is presented along with the various challenges faced by a number of researchers. The difficulty of making progress on these challenges made a lot of researchers shift their attention to other areas that can bring more direct results.

2.2 Story understanding systems

Several systems have been developed to date, to deal with the problem of story understanding and text comprehension. The majority of the developed systems are based on a symbolic representation of the story or script. These systems follow similar architectures in terms of how stories are encoded symbolically, how the background knowledge is encoded, and how the reasoning engine operates. There are also some attempts to employ different methods like deep learning (Zhang *et al.* 2015) and machine learning (Ng *et al.* 2000). Story comprehension is typically tested through question answering, where a machine reads a passage of text and gives answers to questions. Through this test, the accuracy of the system is measured.

https://doi.org/10.1017/S1471068418000443 Published online by Cambridge University Press

In the 1970s, Charniak presented two systems. In his early work (Charniak 1972), a model was presented for answering questions related to children's stories by relating them to real-world background knowledge. The model used an internal representation language for the story and required an expert user to encode it. In his later work (Charniak 1977), a program called Ms Malaprop was presented, for answering questions posed in a simple story. A semantic representation was used for encoding stories, questions, and answers in the program. The program used commonsense knowledge captured in the frame representation of "mundane" painting.

There is also work on Deep Read (Hirschman et al. 1999), an automated reading comprehension system that accepts stories and answers questions about them. Deep Read's creators used a corpus to conduct experiments using questions on stories with known answers. The system uses pattern matching techniques enhanced with automated linguistic processing. The system responds with a sentence that contains the correct answer in 30–40% of the cases.

Another approach to story comprehension was Quarc (Riloff and Thelen 2000), a rule-based system that reads a story and finds the sentence that best answers a given question. The system uses reading comprehension tests with questions on who, what, when, where, and why. Quarc (QUestion Answering for Reading Comprehension) uses lexical and semantic heuristics to look for evidence that a sentence contains the answer to a question. The rules used by the system are hand-crafted.

Similar to Deep Read is the work of Wellner *et al.* (2006) on the abduction-based comprehension (ABC) system. The system reads a text passage and answers test questions with short answer phrases as responses. It uses an abductive inference engine which allows first-order logical representation of relations between entities and events in the text and rules to perform inference over such relations. The system is also able to report on the types of inferences made while reasoning, allowing it to provide insights on where it is not performing well and give indications on where existing knowledge needs update or where new knowledge is required. The authors report an accuracy of 35% using a strict evaluation metric.

More recent attempts include work by Mueller (2007) on a system which models space and time in narratives about restaurants. In particular, Mueller's system converts narrative texts into templates with information on the dialogues that take place in a restaurant, it uses these templates to construct commonsense reasoning problems, and then it uses commonsense reasoning and the created commonsense knowledge to build models of the dining episodes. Using these models, it generates questions and answers to the questions posed. The system was evaluated on stories retrieved from the Web and from the Project Gutenberg (`https://www.gutenberg.org/`). The evaluation showed that much work is needed for the system to produce highly accurate models.

A relatively new system is Genesis (Winston 2015), which deals with both story understanding and storytelling. It models and explores aspects of story understanding using stories drawn from sources ranging from fairy tales to Shakespeare's plays. The system uses the START parser (Katz 1997) to translate English into a language of relations and events that the system can manipulate. This system is deployed using the Java WebStart mechanism.

Story understanding systems are not limited to text comprehension, but there is also work for other media like comic books that combine textual content with pictures. In work by Iyyer *et al.* (2016), deep neural architectures are tested on cloze-style tasks and the

https://doi.org/10.1017/S1471068418000443 Published online by Cambridge University Press

results show that text comprehension and image comprehension alone are not enough for a machine to comprehend a comic book story.

In terms of technical skills and expertise needed to use the aforementioned systems, the majority of them use a command line interface (CLI) and require users to prepare input files (e.g. story and background knowledge rules) using external tools. The output of these systems is generally in textual form, which makes it difficult for inspecting the resulting model. An exception to this is the Genesis system, which has a visual interface and provides a visual way to represent the output, but it is still a stand-alone application that requires installation on the user's device.

The lack of a visual online environment makes it harder for non-experts to set up and use these systems without prior programming knowledge and explicit knowledge of the specific system's internal mechanisms and representation. 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 (e.g. code folding and syntax highlighting).

2.3 Narratives and explanation

The notion of story understanding is closely related to the notion of explanation. There has been an interesting debate and work on this connection, mostly from the philosophical point of view. Velleman (2003) states that "A story does more than recount events; it recounts events in a way that renders them intelligible, thus conveying not just information, but also understanding. We might therefore be tempted to describe narrative as a genre of explanation." Carroll (2001) describes a narrative as a common form of explanation since it is usual to use narratives to explain how things happened. This is also connected to the causal relations of the events in a narrative. Forster (2010) uses the term "plot" to describe a story that is distinguished by the "why?" question and to separate it from one that is connected with the "and then?" question. The first is a form of explanation since one needs to answer the "why" question that includes a causal link between the story concepts. The work of Roth (1989) includes a discussion on whether narratives explain.

Other work concentrates on explanation and scientific understanding (Friedman 1974) and tries to combine the use of narratives with explanation of scientific notions with the purpose to answer why and how questions. Recent work by Morgan (2017) investigates the role of narratives in the social science case-based research, by creating a productive ordering of the materials within such cases, and on how such ordering functions in relation to "narrative explanation."

The Web-STAR IDE allows the encoding of both causal rules in the background knowledge and questions in the story that provide explanations on the story concepts. Moreover, the debugging options offered by the Web-STAR IDE provide in-depth insights on the inferences made to provide answers to questions and hence lead to the relevant explanation.

2.4 Web-based IDEs

Web-based IDEs are systems available through a web browser with no reliance on specific hardware or software stack and are agnostic to the Operating System. Some of these are

https://doi.org/10.1017/S1471068418000443 Published online by Cambridge University Press

now considered as mainstream IDEs for developing applications, such as Cloud9 (Cloud9 IDE 2017), Codiad (Safranski 2017), ICEcoder (ICEcoder Ltd 2017), Codeanywhere (Codeanywhere Inc. 2017), and Eclipse Che (Eclipse Foundation 2017).

These web-based IDEs allow users to write code in an online source code editor using the programming language of their 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.

Web-based IDEs are also used in the logic programming domain. There are only a few systems developed to address this need, like SWI-Prolog for Sharing (SWISH) (Wielemaker *et al.* 2015), Imperative Declarative Programming Web-IDE (IDP Web-IDE) (Dasseville and Janssens 2015), and answer set programming (ASP)-specific IDEs and tools, like the system presented in the work of Marcopoulos *et al.* (2017).

SWISH is a web front-end for SWI-Prolog and is used to run small Prolog programs for demonstration, experimentation, and education. The platform offers collaborative tools for users to share programs with others, and a chat functionality. An instantiation of the system was also used to build the SWISH DataLab system (Bogaard et al. 2017), which is oriented toward data analysis.

Additionally, there is also a web-based implementation of the computer language Logic-based Production System (LPS) (Kowalski and Sadri 2016) built as an extension of SWISH. This system aims at supporting the teaching of computing and logic in secondary school.

The SWISH design is geared toward the educational domain, allowing learners of the Prolog language to easily access code examples and execute them without the need to install SWI-Prolog locally. It exposes only a limited subset of the SWI-Prolog language, and it is not recommended for large and real-world applications.

The IDP Web-IDE is an online front-end for IDP, a Knowledge Base System for the FO(·) language. FO(·) is an extension of first-order logic (FO) with types, aggregates, inductive definitions, bounded arithmetic, and partial functions (Denecker and Ternovska 2008). The Web-IDE allows users to open a chapter from the online tutorial and start testing example programs. There are options for collaborative work and visualization functionality for some of the program outputs.

In the work of Marcopoulos *et al.*, an online system is presented with a cloud file system and a simple interface, which allows users to write logic programs in the SPARC language (Balai *et al.* 2013) and perform several tasks over the programs. The authors aim to use this system to teach ASP to undergraduate university students and high school students.

3 The STAR system

Based on the successful use of web-based IDEs in logic-based systems, we present in this work the development of a web-based IDE for the STAR system. The STAR system is based on the well-established argumentation theory in artificial intelligence (Baroni *et al.* 2011; Bench-Capon and Dunne 2007), uniformly applied to reason about actions and change in the presence of default background knowledge (Diakidoy *et al.* 2015). The STAR system follows guidelines from the psychology of comprehension, both for its

https://doi.org/10.1017/S1471068418000443 Published online by Cambridge University Press

```
1  Bob called Mary on the phone.
2
3  Was Mary embarrassed?
4  Was the phone ringing?
5
6  She did not want to answer the phone.
7  Bob had asked her for a favor.
8  She had agreed to do the favor.
9
10 Was the phone ringing?
11
12 She answered the phone.
13 She apologized to Bob.
14
15 Was Mary embarrassed?
```

Fig. 1. A short story in natural language with interspersed questions.

representation language and for its computational mechanisms for building and revising a comprehension model as the story unfolds.

In terms of its underlying infrastructure, the STAR system is written in SWI-Prolog (Wielemaker *et al.* 2012). Upon the setting up of the Prolog environment, and the invocation of the system, a user-selected domain file is loaded and processed. We present the syntax and semantics of the STAR system through the example story in Figure 1.

The example story is interspersed with questions. These are not meant to be parts of the story, but are questions directed toward the reader of the story. Whenever a sequence of questions is encountered, the reader is expected to provide answers to the questions based on the information given in the story in all the preceding story lines. The story then continues until a new sequence of questions is encountered, and so on. Each such part of the story is effectively a *scene* or a reading *session*, and each session is associated with the questions that need to be answered based on the information provided in that and all preceding story sessions. Although the reader goes through the story in the linear fashion in which the story is represented, the story time need not be linear, and can jump back and forth between different time periods. Questions are assumed to refer to a story time-point following the one at which the last story session left off.

As the story unfolds, answers to questions might change either because the same question is asked at a different point in the story time-line, or because the story information leads the reader to revise their *comprehension model* of what they infer (based on their background knowledge and the given story information) to be the case in the story world. The two questions in the example have their answers changed as a reader progresses from the top to the bottom of the story, with the question "Was the phone ringing?" changing because of the first of the aforementioned reasons, and the question "Was Mary embarrassed?" changing because of the second of the aforementioned reasons.

Having explained how a reader may comprehend our example story, we hasten to note that the STAR system *does not* process stories in natural language – in fact, processing stories in natural language is one of the main features of the web-based IDE that is presented in this work. Instead, the STAR system expects the story statements, their partitioning into sessions, and their association with questions to be provided in a certain

https://doi.org/10.1017/S1471068418000443 Published online by Cambridge University Press

```
1  session(s(0),[],all).
2  session(s(1),[q(1),q(2)],all).
3  session(s(2),[q(3)],all).
4  session(s(3),[q(4)],all).
5
6  s(0) :: is_favor(favor1) at always.
7  s(0) :: is_person(bob) at always.
8  s(0) :: is_person(mary) at always.
9  s(0) :: is_phone(phone1) at always.
10
11 s(1) :: call(bob, mary, phone1) at 6.
12 s(2) :: -do_want(mary,answer(phone1)) at 12.
13 s(2) :: have_ask(bob, mary, favor1) at 2.
14 s(2) :: have_agreed(mary,do(favor1)) at 4.
15 s(3) :: answer(mary, phone1) at 16.
16 s(3) :: apologize(mary, bob) at 18.
17
18 q(1) ?? is_embarrassed(mary) at 8.
19 q(2) ?? is_ringing(phone1) at 10.
20 q(3) ?? is_ringing(phone1) at 14.
21 q(4) ?? is_embarrassed(mary) at 20.
```

Fig. 2. A possible representation of the example story depicted in Figure 1 to the STAR syntax.

symbolic language. All these elements constitute the first part of the domain file that the STAR system loads once it is invoked. A possible representation (although by no means the only one) of our example story is given in Figure 2.

Each of the story statements is of the form `s(N) :: Literal at Time-Point`, where `N` is a non-negative integer representing the session of that statement; session `0` is a special session that includes typing information only. 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. The representation of our example story clearly shows its nonlinear time-line.

Following the story statements are the question statements of the form `q(N) ?? Literal at Time-Point; Literal at Time-Point; ....`, where `N` is a non-negative integer representing the number of the question and ";" separates the possible answers to that question; although the notation is meant to represent multiple-choice questions, in effect the STAR system treats each of the choices as a true/false question. Which questions are associated with which sessions is given by the session statements.

Given the story and question representation in Figure 2, the STAR system aims to produce a comprehension model of the story, through which it will subsequently attempt to answer the posed questions. Much like human readers, the STAR system invokes background knowledge about the story world to infer what else holds beyond what is explicitly stated in the story. This background knowledge is also represented in a logic-based language and constitutes the second part of the domain file. For our example story, and in a manner consistent with our chosen symbolic representation of that story, a possible representation of (some of) the background knowledge relevant for the story is given in Figure 3.

https://doi.org/10.1017/S1471068418000443 Published online by Cambridge University Press

```
fluents([
   do_want(_,_),
   is_embarrassed(_),
   carried_out(_),
   has_asked_for(_,_,_),
   has_agreed_to(_,_),
   is_ringing(_)
]).

c(01) :: have_ask(P1,P2,S) causes has_asked_for(P1,P2,S).
c(02) :: have_agreed(P2,do(S)) causes has_agreed_to(P2,S).

p(11) :: has_asked_for(P1,P2,S), has_agreed_to(P2,S), apologize(P2,P1)
         implies -carried_out(S).

c(21) :: have_agreed(P2,do(S)), -carried_out(S) causes is_embarrassed(P2).

c(31) :: has_asked_for(P1,P2,S), has_agreed_to(P2,S), -carried_out(S),
         call(P1,P2,D), is_phone(D) causes -do_want(P2,answer(D)).

c(41) :: is_person(P1),is_person(P2),call(P1,P2,D),is_phone(D) causes
         is_ringing(D).

c(42) :: is_person(P1),answer(P1,D),is_phone(D) causes -is_ringing(D).

c(42) >> c(41).
```

Fig. 3. A possible representation of the background knowledge for comprehending the story depicted in Figure 1, assuming its encoding in Figure 2.

The presented representation includes four type of statements[1]: a list of concepts that are marked as *fluents*, indicating that their truth value persists across the story time-line; rules prefixed by the symbols `c(N)` and `p(N)` to indicate that they are, respectively, causal or property rules, and priorities `>>` indicating relative strength between conflicting rules.

The main part of a rule is of the form `Body causes / implies Head.`, where `Body` is either the tautology `true`, or a comma-separated list of literals, and `Head` is a single literal. Each rule, then, expresses an implication from the premises in its body to the conclusion in its head. The difference between the causal and property rules lies in their treatment of time. Property rules are meant to capture dependencies between the properties of entities, and refer to any single point in the story time-line: whenever the body holds, the head also holds at that same time-point. On the other hand, causal rules are meant to express how things change over time, and capture dependencies between consecutive time-points: whenever the body holds, the head holds at the following time-point. Further, when the head literal of a causal rule is inferred, this inference *causes* the persistence of the truth-value of that literal from earlier time-points to stop in case the persisted truth-value conflicts with the inference. In effect, the fluent list expresses implicitly a third type of persistence rules, along with the implicit lower priority compared to all conflicting causal rules. Additional priorities between causal and/or property rules are expressed explicitly.

[1] The STAR syntax allows additional types of statements and expressivity, which we do not present here for simplicity.

https://doi.org/10.1017/S1471068418000443 Published online by Cambridge University Press

We will not discuss in detail the intuitive interpretation of the rules in the background knowledge for our example story, other than to say that they roughly capture the knowledge that: if you apologize to someone that has asked you to do something, to which you have agreed, then it is because you have not carried out that something; having agreed to do something that you have not carried out causes embarrassment, and further causes not wanting to answer the phone when the call is from the person that has asked you for that something; a call causes the phone to start ringing, and answering the phone causes the ringing to stop.

With all the aforementioned logic-based information in a domain file, the STAR system proceeds to construct a comprehension model of the story. A comprehension model can be thought as a partial mapping from timed concepts to truth-values, essentially indicating when each concept is true, false, or unknown. In computing these truth-values, one takes into account both the information given explicitly in the story, but also draws inferences through the background knowledge. The STAR system adopts a particular argumentation-based approach to how inferences are drawn. Roughly, it combines story statements with rules to build a proof of the entailment of a literal. Rules are used both in the forward direction (i.e. via modus ponens) and in the backward direction (i.e. via modus tolens). Since different combinations of story statements and rules might lead to contradictory inferences, each constructed proof is viewed as an argument in support of an inference, and conflicts between arguments are resolved by lifting the priority relation between rules to an attacking relation between arguments (Rahwan and Simari 2009).

Without discussing the technical nuances of the particular argumentation approach that the STAR system adopts (Diakidoy *et al.* 2014), we briefly comment on how the STAR semantics relate to existing well-established argumentation semantics. STAR adopts a structured rule-based argumentation framework in the spirit of ASPIC+ Combinations of premises from the story with defeasible rules from the background knowledge form a proof tree in support of some inference; this tree corresponds to an argument. Unlike in the Assumption-Based Argumentation (ABA) framework (Toni 2014) the premises are assumed to be indefeasible, and (exogenously) attack any argument that supports a contrary argument that supports a contrary inference. Also unlike in the ABA framework, arguments (endogenously) attack each other on the rules they use, not on their premises. An attack comes from the last / head / top rule in the proof tree of an argument, and is directed toward any (possibly internal) rule in the proof tree of another argument. As long as the former rule is not less preferred than the latter rule, the attack is present. The semantics of the attack relation implies, in particular, that a pair of arguments can attack each other. With this attack relation, the STAR system proceeds to compute the grounded extension of the resulting argumentation framework, and offers this unique extension as the comprehension model of the story. Beyond consulting the relevant work for more details (Diakidoy *et al.* 2014), the interested reader may wish to also consult a more recent work (Michael 2017), where a similar in spirit argumentation semantics is discussed, without the nuances of temporal reasoning and contrapositive reasoning, and where a case is also made for the learnability of this type of arguments.

Once the grounded extension is computed after each session (with the arguments that are relevant given the premises of the story that far), the STAR system outputs the computed comprehension model, as in Figure 4 for our example story.

The output presents the comprehension model (literals that are true at each timepoint), with parts of it marked in triangular parentheses to indicate that those come

https://doi.org/10.1017/S1471068418000443 Published online by Cambridge University Press

```
1  ==================================
2  >>> Reading story up to scene s(3)
3  ==================================
4  >>> Universal argument...
5  >>> Acceptable argument...
6
7  >>> Comprehension model:
8
9  0: -carried_out(favor1) < is_favor(favor1)> < is_person(bob)>
10    < is_person(mary)> < is_phone(phone1)> < is_ringing(ringing1)>
11    < is_ringing(ringing2)>
12
13 1: -carried_out(favor1) < is_favor(favor1)> < is_person(bob)>
14    < is_person(mary)> < is_phone(phone1)> < is_ringing(ringing1)>
15    < is_ringing(ringing2)>
16
17 2: -carried_out(favor1) < is_favor(favor1)> < is_person(bob)>
18    < is_person(mary)> < is_phone(phone1)> < is_ringing(ringing1)>
19    < is_ringing(ringing2)> < have_ask(bob,mary,favor1)>
20
21 ...
22
23 18: -carried_out(favor1) is_embarrassed(mary) < is_favor(favor1)>
24     < is_person(bob)> < is_person(mary)> < is_phone(phone1)>
25     -is_ringing(phone1) < is_ringing(ringing1)> < is_ringing(ringing2)>
26     < apologize(mary,bob)> -do_want(mary,answer(phone1))
27     has_agreed_to(mary,favor1) -call(bob,bob,phone1)
28     -call(bob,mary,phone1) -call(mary,bob,phone1) -call(mary,mary,phone1)
29     has_asked_for(bob,mary,favor1)
30
31 19: -carried_out(favor1) is_embarrassed(mary) < is_favor(favor1)>
32     < is_person(bob)> < is_person(mary)> < is_phone(phone1)>
33     -is_ringing(phone1) < is_ringing(ringing1)> < is_ringing(ringing2)>
34     -do_want(mary,answer(phone1)) has_agreed_to(mary,favor1)
35     -call(bob,bob,phone1) -call(bob,mary,phone1) -call(mary,bob,phone1)
36     -call(mary,mary,phone1) has_asked_for(bob,mary,favor1)
37
38 20: -carried_out(favor1) is_embarrassed(mary) < is_favor(favor1)>
39     < is_person(bob)> < is_person(mary)> < is_phone(phone1)>
40     -is_ringing(phone1) < is_ringing(ringing1)> < is_ringing(ringing2)>
41     -do_want(mary,answer(phone1)) has_agreed_to(mary,favor1)
42     has_asked_for(bob,mary,favor1)
43
44 >>> Answering question q(4):
45 + accepted choice: ,[is_embarrassed(mary)at 20]
46
47 >>> Finished reading the story!
```

Fig. 4. Part of the output of the STAR system for the story depicted in Figure 1, as encoded in Figure 2 and with the associated background knowledge presented in Figure 3.

directly from the story and not from inferences. Following the comprehension model, each of the questions (of the session being processed) is presented along with all their choices for answers, and for each answer the system responds on whether it is accepted, rejected, or possible, depending on whether the answer appears affirmatively, appears negatively, or is absent in the comprehension model.

Additionally to the above, the user may also select to present only part of the comprehension model, or have the system present the arguments that it used to support the inferences that led to the comprehension model. Roughly, the user may request to see: the

https://doi.org/10.1017/S1471068418000443 Published online by Cambridge University Press

"universal argument" showing all rules that are activated by the story (in all extensions) without regard to conflicts, the "acceptable argument" showing those activated rules that are accepted (in the grounded extension) after the argumentation semantics resolve all conflicts, and details on which rules are qualified ("attacked") by other rules to help the user understand why certain rules did not end up in the acceptable argument. Since the comprehension model is revised from session to session, the user may also see which rules become obsolete and are retracted across sessions (i.e. part of the grounded extension for the preceding but not the current session), and which new rules come into play and are used to elaborate the comprehension model (i.e. part of the grounded extension for the current but not the preceding session). Finally, the user may choose to see how much time the STAR system spends in each part of its computation, and to decide whether the relevant part of the story will be shown along with each session.

More details about the STAR system can be found in the work of Diakidoy *et al.* (2015) and on the system's website (`http://cognition.ouc.ac.cy/star`). The full example used in this section is available from the "Help" menu of the Web-STAR IDE.

4 Building Web-STAR on top of STAR

Following the successful paradigm of many other projects that moved to an online environment, and aiming toward increasing the usage of the STAR system from non-expert users, we developed a web-based IDE for STAR. This IDE incorporates all the functionality of the STAR system in a structured web environment with the addition of visualization, automation, and collaboration tools that help users prepare and process their stories.

Moreover, the IDE employs a number of social features for user collaboration, like public code sharing and posting of stories to a public repository. In addition, users can work together using a state-of-the-art collaboration component which 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, it is OS agnostic, and offers modern IDE functionalities;
- a platform for collaboration and educational support;
- a platform for integrating story comprehension functionality to other systems; and
- a modular architecture that facilitates the addition of new components and functionality.

Figure 5 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, a presentation of the web-based IDE is shown with details of the workspace layout, the components, and the functionality of the IDE. The Web-STAR IDE is available online at `http://cognition.ouc.ac.cy/webstar/` and it is accessible from any device.

4.1 Getting started with the Web-STAR interface

To start using the Web-STAR IDE, a user creates an account and activates the personal workspace. Currently, both local and remote authentication options are available.

https://doi.org/10.1017/S1471068418000443 Published online by Cambridge University Press

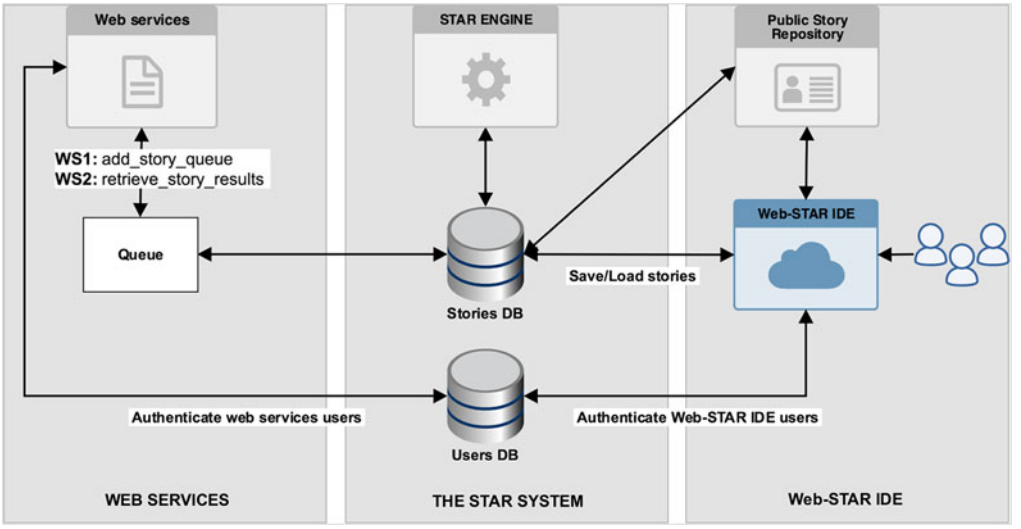


Fig. 5. The Web-STAR platform architecture with its three core components: the Web-STAR IDE, the STAR system, and the web services infrastructure. The diagram also presents the authentication mechanism, the storage functionality, and the web services provided.

The local authentication method uses the integrated storage facilities of the platform. The remote authentication method uses the OAuth2 protocol (https://oauth.net/2/), offered by third parties like Facebook, Google, Github, so on. Other authentication methods are supported as long as the appropriate plugin is available.

After the authentication process is completed, the user is redirected to the Web-STAR IDE environment where both the source code editors and the visual editors are present.

4.2 The IDE environment and workspace

Users are presented with the workspace (see Figure 6), which is divided into three distinct areas: (i) the story writing area, (ii) the background knowledge writing area, and (iii) the story comprehension output area. This design was chosen to give users a clear understanding of the workflow of the story comprehension process, and to enable users to hide the areas which are not needed, aiming to avoid information overload.

The workspace is also divided into two columns. The left column is for the tasks that do not require users to have prior knowledge and experience in using the STAR system and the right column is for more seasoned users who have prior knowledge of the STAR system semantics and experience in encoding stories using the STAR system. This modular layout allows users to choose the mode they want to use while preparing their stories. More specifically, the web interface comprises three view modes:

- **Simple**: Users write a story in natural language, add background knowledge using the visual editor, and view the visual representation of the story comprehension model. This mode is ideal for users that are new to story understanding systems and want to have an overview of the capabilities and processes of encoding a story.
- **Advanced**: Users write a story in the STAR syntax, encode the background knowledge in the source code editor, and are presented with raw output from the STAR reasoning engine. This mode is ideal for users with prior knowledge in encoding

https://doi.org/10.1017/S1471068418000443 Published online by Cambridge University Press

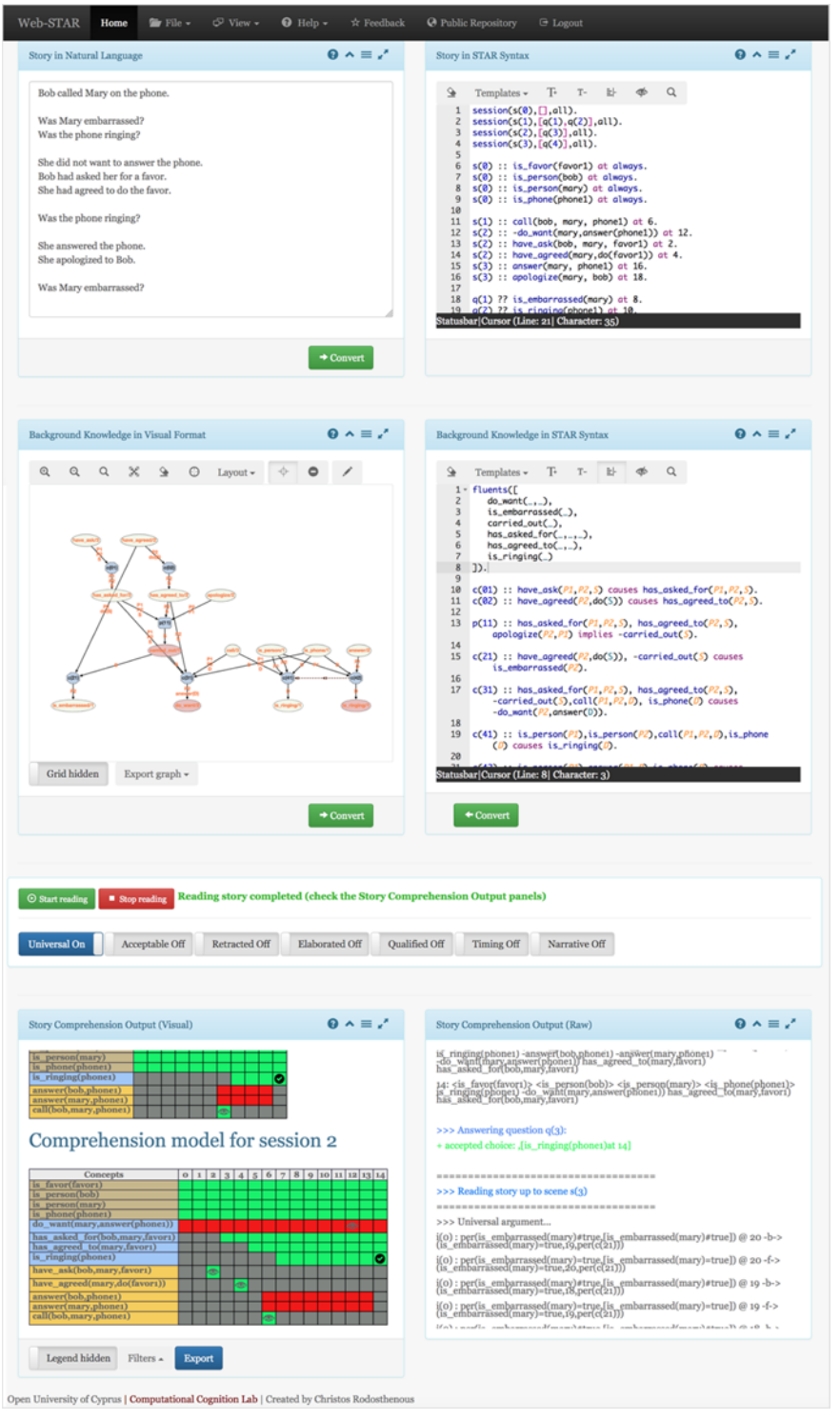


Fig. 6. A screenshot of the Web-STAR IDE layout. The workarea is divided into two columns: the left column (Simple mode) and the right column (Advanced mode); and three rows: the story area, the background knowledge area, and the story comprehension output area.

https://doi.org/10.1017/S1471068418000443 Published online by Cambridge University Press

stories in the STAR syntax, and for users who wish to enter pre-encoded stories as done in the stand-alone version of the STAR system.

- **Mixed**: Users write a story using any of the above options and convert from one mode to the other. For example, users can encode the background knowledge using the source code editor and then convert it to the visual format where they can make further changes. This mode is ideal for users that are learning the system and feel more confident in using the visual components and viewing the conversion. Moreover, this mode is used for teaching, since educators can present examples of encoding the story in visual format and then present the corresponding STAR syntax.

Additionally, users are also able to set the active area in any of the above view modes, that is, to display only the background knowledge area, the story input area, or the story comprehension output area. The design of the workarea is fully customizable, allowing users to maximize the part of the IDE they are currently working on and minimize the areas that are not needed at that time. Whenever a mode is chosen, the relevant area is resized to maximize the view to the screen size of the device.

4.3 The story workarea

Users can start using the system by creating a story from scratch, either in natural language or in the STAR syntax, or even by loading an existing story. More specifically, users can write their code in the source code editor or load it from an external file previously created for the stand-alone STAR system, load an example file, or load a story file from the public repository. Non-experts can benefit from the example stories and the user-contributed stories in the public repository.

Currently, the source code editor (see Figure 7) allows syntax highlighting using a STAR syntax highlighter file that inherits the Prolog's syntax template and is expanded with the STAR semantics. Furthermore, line numbering and code wrapping are also available to users along with the extensive "search & replace" capability for finding text in large stories.

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 stories and a public workspace for loading other users' stories. A story that is saved in the personal workspace can only be accessed by its creator, whereas a story 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 the stand-alone version of the STAR system to process the story.

When a user loads a story, like the example story presented in Section 3, the source code editor immediately identifies and highlights the STAR semantics (variables, rules, and operators) and makes it easier for the user to read the encoded story (see Figure 7). After studying the file, the user can move to the questions part and can add one or more questions by choosing the "question template" from the menu. The question template (presented in Section 3) is added and the user can add the predicates and time-points at which the question is posed. When changes are made to the example file, the user can save it to the personal workspace using the corresponding menu option.

https://doi.org/10.1017/S1471068418000443 Published online by Cambridge University Press

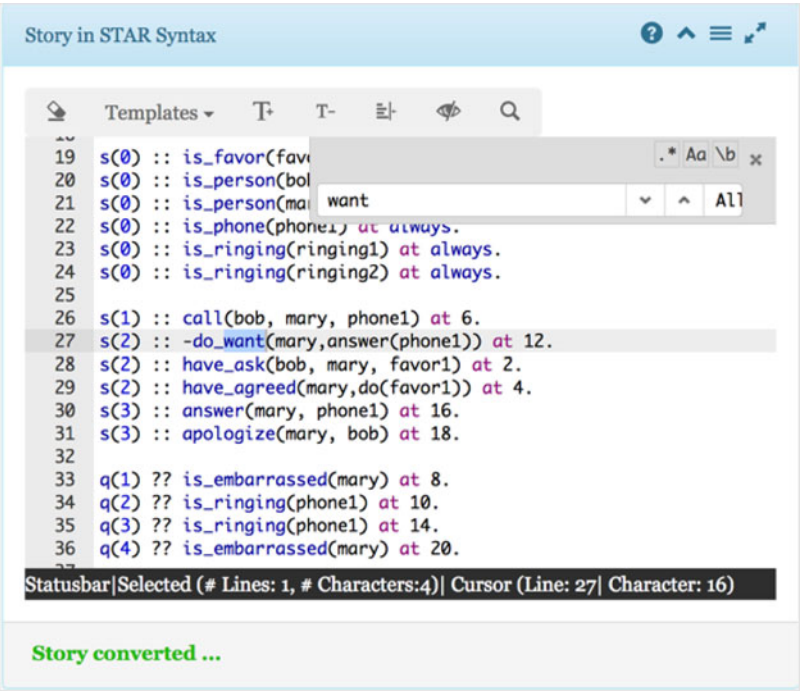


Fig. 7. A screenshot of the source code editor, depicting the line numbering, syntax highlighting, and line highlighting functionality. Above the source code editor resides the toolbar menu with the "search & replace" functionality window open. At the bottom of the editor resides the statusbar with information on the selected line and character.

4.3.1 Natural language to the STAR syntax converter

One of the innovations available to the Web-STAR user is the automated component to convert a story from natural language to the STAR syntax. This is a real-time process, where the story and the questions are written in natural language, and the system processes them using a natural language processing (NLP) system and a custom-built parser that maps processed words and phrases to predicates with their arguments.

More specifically, each sentence is processed using the Stanford CoreNLP (Manning *et al.* 2014) system for named entity recognition (NER), part-of-speech, lemmas (canonical or base form of the word), basic dependencies, and coreference resolution.

First, the component automatically identifies the sessions or scenes of the story. Sessions are added when a series of statements are followed by a question, or a group of questions. There is always a base session "**Session S(0)**" where all the constants are represented. For each group of questions, an additional session is created (e.g. see the story in Section 3, Figure 2).

Next, the nouns in each sentence and the named entity types (location, person, organization, money, percent, date, and time) are identified to create the concepts that represent constant types. For each named entity, a statement of the form "`is_<EntityType> (<entity>) at always.`" is added to the base session of the story. Personal pronouns are also identified as a `Person` entity. The following is an example of this: `is_person`

https://doi.org/10.1017/S1471068418000443 Published online by Cambridge University Press

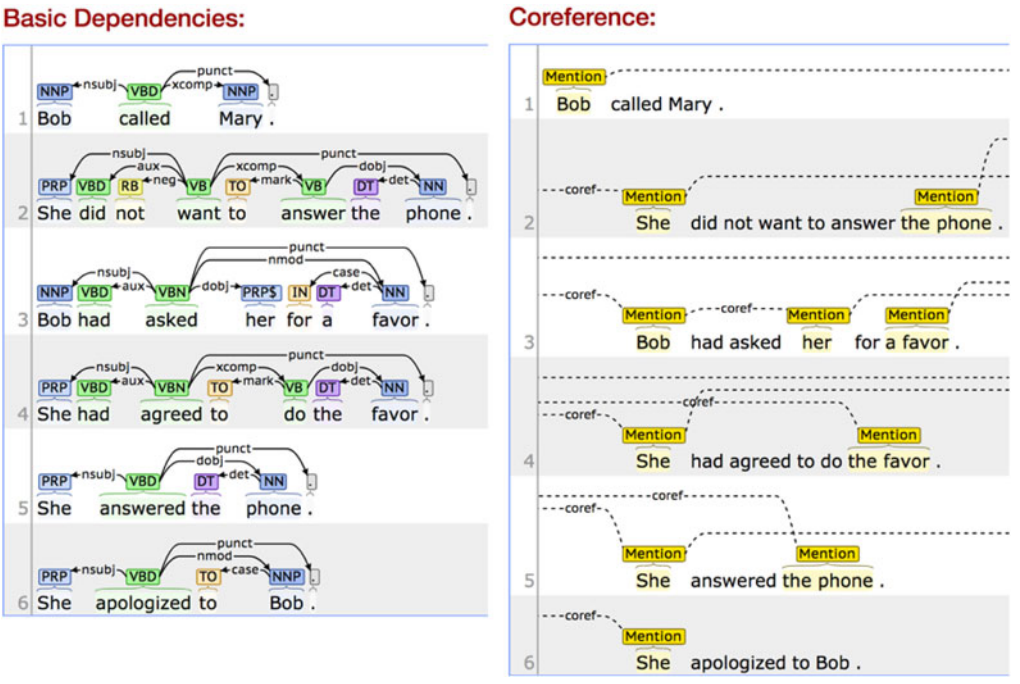


Fig. 8. The output of the CoreNLP processing for the example story in Figure 1. On the left side, the basic dependencies are presented in graphical form and on the right side the coreferences are depicted.

`(personX) at always.`, where X is an integer, representing the number of entities with the same name. When a coreference is found, the identified person name is used in the concept.

Predicates are created using the Stanford basic dependencies for extracting textual relations from the text. More specifically, for each sentence, the lemmatized "`ROOT`" is used as the predicate name and the lemmatized text from types "nsubjpass, dobj, nmod:poss, xcomp" is used to create the predicate arguments in lower case.

When a word is characterized with the "aux" or "compound:prt" types, then the predicate name is expanded with a "_" and the new word is added in front of the predicate name. When a word has a "neg" type, then the negation symbol "-" is added in front of the predicate name. Words with types "amod, case, cop, auxpass, aux and compound:prt" that are dependent of the "ROOT" are also appended to the predicate name. For words with types "aux, aux:pass and cop" when the lemma "be" is identified, it is converted to "is".

For adding time-points to the story statements, we start at time-point 2 and form two lists. First a list with statements in past perfect is formed starting from time-point 2 and increasing by two for each statement. Then a list with statements starting from the maximum time-point of the first list with an increment of two is formed. These two lists are joined and form the story statements (see Figure 2).

To better understand the conversion process, we take the example of the story in Figure 1, its representation in the STAR syntax (see Figure 2) and the output from the

https://doi.org/10.1017/S1471068418000443 Published online by Cambridge University Press

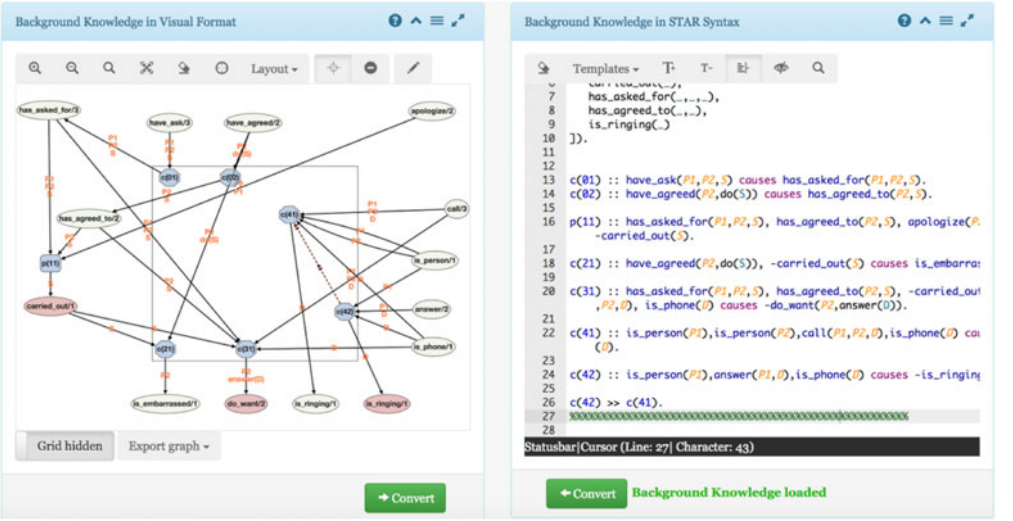


Fig. 9. The background knowledge workarea of the IDE. The visual editor is depicted on the left side of the screenshot, and the source code editor on the right side. There are conversion buttons from one form to the other at the bottom of each panel.

Stanford CoreNLP depicted in Figure 8. In particular, for the sentence: "She had agreed to do the favor" (sentence 4 in Figure 8) we take the ROOT ("agreed") and the dependent words "She","do" and form the predicate `agree (She,do)`. Next, the "ROOT" is connected with an "aux" type with the word "had" and the predicate name is updated accordingly `have_agreed(she,do)`. The word "She" refers to "mary" (using the coreference parsing) and "do" is connected with an "xcomp" relation with the `ROOT`, so "do" will form a new predicate `do(favor1)` and the final concept will become `have_agreed(mary, do(favor1))`.

4.4 The background knowledge workarea

The next step in preparing the story is the encoding of the background knowledge. This can be done either by using the source code editor or the visual editor (see the left side of Figure 9). The visual editor uses a directed graph to represent rules. Users are able to see how rules build on each other (i.e. rules whose body literals are the head literals of other rules) and better understand the reasoning process. Moreover, users can focus on specific rules and literals and understand their role in forming the comprehension model of the story.

In particular, each rule is represented with a blue-colored node of octagonal shape for causal rules, and of a rounded orthogonal shape for property rules. Literals are represented with nodes of a cyclical shape, and are green or red to indicate that the literal is, respectively, positive or negative. Literals with a directed edge toward a rule node represent body literals for that rule, and the single literal with a directed edge from a rule node represents the head literal for that rule.

Each node is labeled with the rule's or the predicate's name. For literals, the name is created using the predicate's name and the arity of the predicate (e.g. the predicate "have

https://doi.org/10.1017/S1471068418000443 Published online by Cambridge University Press

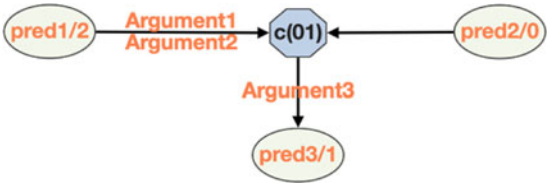


Fig. 10. A visual representation of the rule: `c(01) :: pred1(Argument1,Argument2), pred2 causes pred3(Argument3).`

ask" with three arguments is represented as `have_ask/3`). Arguments are represented with labels on the edges connecting the literal nodes with the rule node (see orange labels on edges in Figure 10).

Users can choose to create the entire background knowledge using the tools of the visual editor. Using the "edit" button, users can add literals, rules, edges, and priorities between rules. More specifically, users choose the desired element and click on the white area of the graph, the "canvas". When a rule is added, the label is automatically set to create a unique name (e.g. `c01, c02, p01, p02`). When literals are added, the user is asked to provide the literal's name, arity, and polarity (positive or negative). Users connect literals with rules using the "edge drawing tool" and can set priorities between rules by drawing a "dashed edge" from one rule node to another. Adding and updating arguments to literals is performed by clicking on the connecting edges between the literal and the rule. A dialog box appears for typing each argument.

For every input (textual or visual) and every action on the canvas that does not conform with the STAR syntax (e.g. having two literals in the head of a rule) a guidance message (not simply an error message) is shown, which explains to the user in a visual way (e.g. by highlighting nodes or edges) what needs to be changed to lift the error. This is helpful in teaching scenarios, where students get to know the environment and the basic semantics of the STAR system (or logic-based programming, more generally).

In cases of stories with a large background knowledge, a user can group rules together and minimize or maximize the view of individual groups, isolating the part of the background knowledge that the user wants to inspect. To support this, a type of code folding functionality is implemented, which allows users to focus on a specific subset of the rules on the screen. In Figure 11, the code folding/unfolding capability of the IDE is presented.

Moreover, users can zoom in and out of the graph and can change its layout dynamically. There are a number of available layouts for users to choose from, like "the circle layout," where nodes are put in a circle, "the breadthfirst layout," where nodes are put in a hierarchy based on a breadthfirst traversal of the graph, so on. Furthermore, users can search for a rule, literal or argument using the search tool. When the element is found, it is maximized and focused along with its neighboring elements. There is also an option for fitting the graph to the screen, as well as a "graph navigator" option that allows users to have a bird's eye view of the whole graph and navigate to the desired part of it.

The Web-STAR IDE allows filtering out elements of the graph that are not needed for a specific job. For example, users can choose to "toggle" the visibility of causal or property rules, priorities, rules with low density, or rules that are not connected with other rules. This functionality is part of the Web-STAR's IDE ability to handle large background knowledge bases with rules.

https://doi.org/10.1017/S1471068418000443 Published online by Cambridge University Press

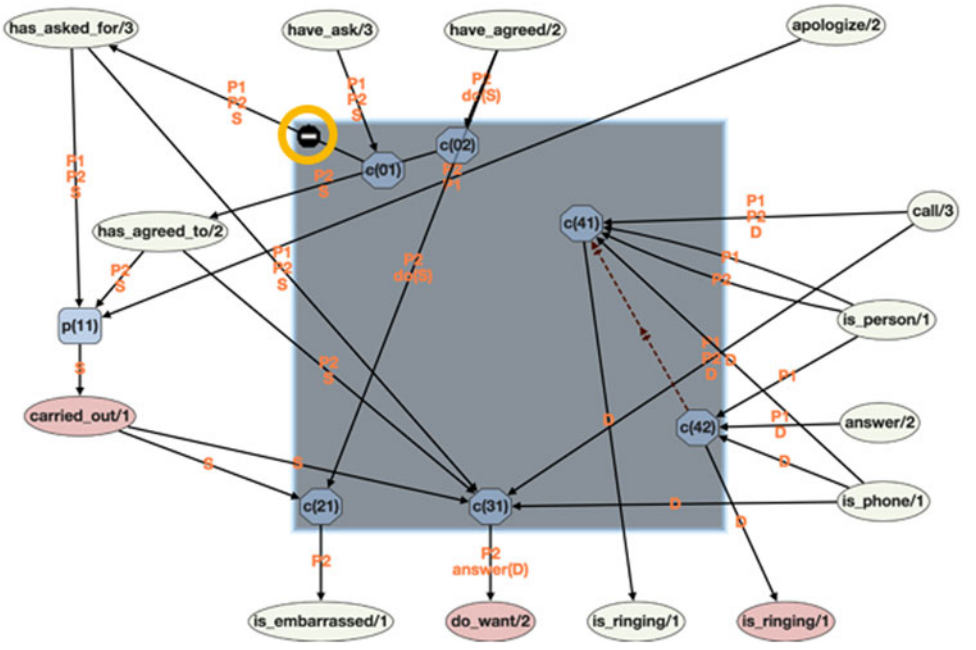


Fig. 11. A screenshot of the folding/unfolding code capability of the Web-STAR IDE. Grouped rules can be maximized and minimized by clicking at the top left corner of the grouping.

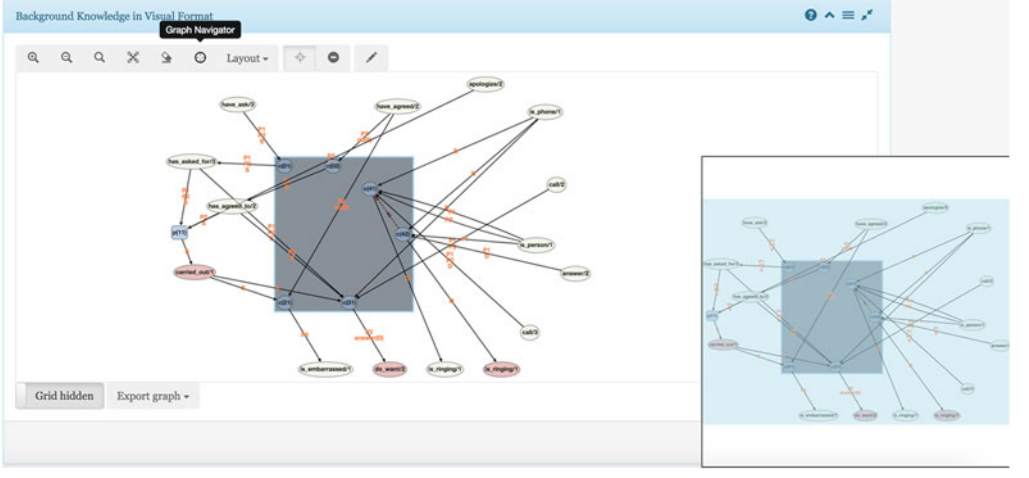


Fig. 12. A screenshot of the "graph navigator" window (bottom right), where users can have a bird's eye view of the background knowledge graph and navigate to the desired part of it.

The background knowledge graph can be exported in various formats, including image formats (png, jpg), JSON, and GraphML (Brandes *et al.* 2013), which can subsequently be used with third-party applications to present or process the graph and its data.

Expert users can use the source code editor in parallel with the visual one. A similar in look and feel editor with the one for preparing stories is available, and users can take advantage of the included templates for adding rules.

https://doi.org/10.1017/S1471068418000443 Published online by Cambridge University Press

4.4.1 Converting background knowledge from visual to textual format

Converting background knowledge from visual to textual format and vice versa, is performed with the click of a button, allowing the user to encode parts of the knowledge in the one format or in the other. Web-STAR's internal mechanisms read each graph element and perform the conversion of visual background knowledge rules to STAR format. Details on this process are provided in Algorithm 1.

Algorithm 1 Convert background knowledge graph to the STAR syntax

```
1: procedure CONVERT_GRAPH_TO_STAR(GRAPH_OBJECT)
2:     % Get all the rule nodes of the graph object
3:     for Each node i do
4:         %Get the edge directed outwards of the node (Head)
5:         for Each edge j1 do
6:             literal=edge[ j1].connected_node
7:             head=convert_predicate_star(literal,edge[ j1].label)
8:             push_to_list_of_head_literals(head)
9:         end for
10:        %Get all the edges directed towards the node (Body)
11:        for Each edge j2 do
12:            literal=edge[ j2].connected_node
13:            body=convert_predicate_star(literal,edge[ j2].label)
14:            push_to_list_of_body_literals(body)
15:        end for
16:        %Proceed and create the textual representation of the rule
17:        if node[i].type===“property” then
18:            rule_type=“implies”
19:        else
20:            rule_type=“causes”
21:        end if
22:        rule=node[i].name :: node[i].body_literals rule_ type node[i].head_literal
23:    end for
24: end procedure
```

4.5 Story comprehension process and output

After completing the story preparation and the background knowledge encoding, users can proceed with the story comprehension process. Users can click the “Start reading” button and immediately see results coming from the STAR system in the “Story Comprehension Output” area in real time. A number of reporting options can activate and expose the internal processes of the STAR system, including the argumentation mechanism applied for story comprehension, for debugging or educational purposes. In particular, users can choose to view all arguments (`Universal`), the subset of acceptable arguments (`Acceptable`), arguments removed during a specific session (`Retracted`), arguments added during a specific session (`Elaborated`), and information about which arguments qualify other arguments (`Qualified`).

https://doi.org/10.1017/S1471068418000443 Published online by Cambridge University Press

When the reading process is completed, users can view both the comprehension model and the answers to questions posed. This can be done both in a visual and textual format. The visual output might be preferred for tracking each concept across the story timeline, and the textual output might be preferred for debugging. Each panel is dynamically updated when new information is sent from the STAR system.

In Figure 13, the visual output of the comprehension model is depicted, presenting the state of each concept at each time-point. Green, red, and dark grey represent concepts whose value is, respectively, positive, negative, or unknown at that time-point.

The magnifying glass marks concepts whose value is observed at that time-point, that is, they are extracted from the narrative directly. 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(bob) at always.`).

Users can apply filters on the output of the comprehension model and focus their attention of particular concepts. They can choose, for instance, to filter out fluents, actions, and constants, or to view only concepts whose value changes through time, concepts that have a high frequency in the background knowledge, or even concepts that are part of causal rules. The latter are a good indication of the focus of the story and its parts that are most interesting to a reader (Goldman *et al.* 1999).

The model can be exported in various formats and can be used for educational purposes. The Web-STAR IDE has also a textual format of the story comprehension output that presents the raw output of the STAR system as it would appear when executed as a stand-alone application. This output is enhanced with color highlighting to identify questions, positive or negative answers to questions, and debugging messages.

4.5.1 Collaboration and "social" options

Apart from the typical IDE functionality, Web-STAR IDE also provides functionality for sharing publicly a story with other users. By clicking the "Share it" button a story is added to the public stories repository (see Figure 14) and appears in the "public stories" tab in the story browser dialog. Users can read shared stories and add comments, supporting the education of new users from more expert ones.

Beyond sharing, users can collaboratively write a story using the collaboration functionality provided. The system produces a link that can be sent to anyone interested in collaborating for a specific session. The recipient of the link can see the screen and the mouse pointer of each participating user, and changes of content in real time, while also being able to chat through text and audio. This setting enables teams to collaborate on preparing a story, and allows students to learn by working together on class projects.

4.5.2 User support and feedback

The Web-STAR IDE offers a number of features to help its users achieve their goals. Firstly, users can follow a guided tour through the Web-STAR IDE features. Users can then start testing the functionality of the platform with the examples available in the story browser. These examples were carefully crafted for teaching the STAR semantics.

https://doi.org/10.1017/S1471068418000443 Published online by Cambridge University Press

https://doi.org/10.1017/S1471068418000443 Published online by Cambridge University Press

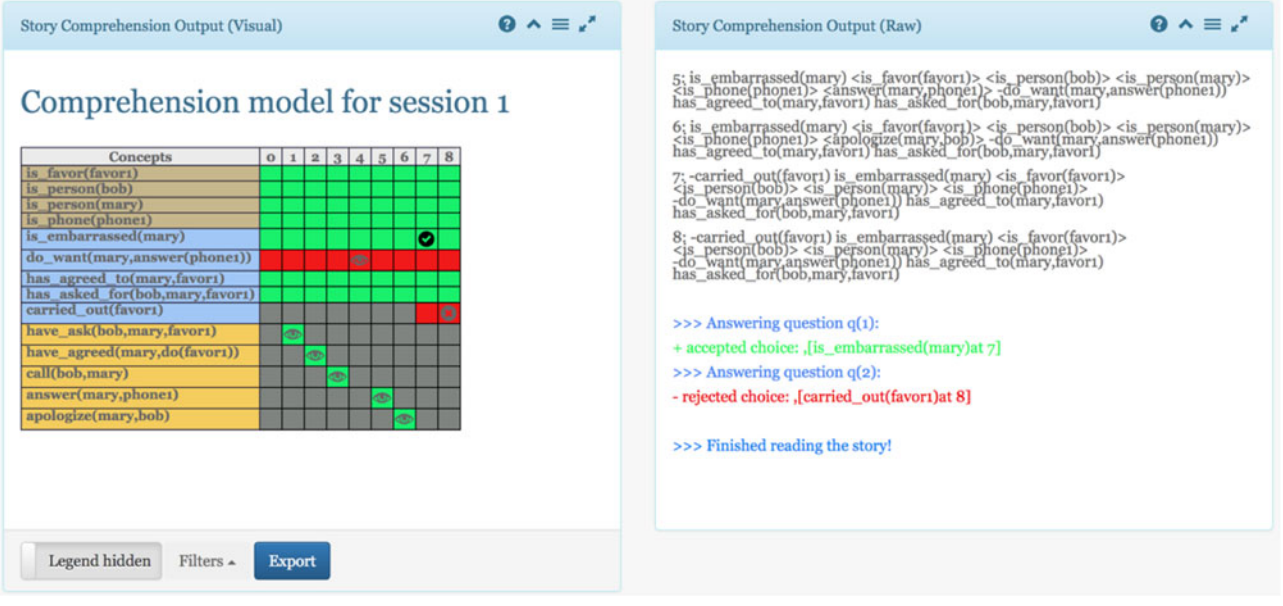


Fig. 13. A screenshot of the "Story comprehension output" workarea of the IDE, depicting the comprehension model. The legend above the comprehension model provides details on the meaning of symbols and colors in the visual representation of the model and its visibility can be toggled by using the relevant switch. On the right side, the raw output for the same story is presented.



Fig. 14. 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.

Moreover, in each panel there is an online help option, for guiding users to the specific functionality available for that panel. The icons and graphics chosen for buttons and toolbars are inline with the ones users are familiar in other IDEs. In cases where some users are not aware of the meaning of an icon, a tooltip is available.

To allow users to provide feedback on new desired functionalities or encountered problems, Web-STAR offers a built-in feedback functionality that stores a user's message in the platform's database and alerts the developers through email.

4.6 Technical details and challenges

For designing and implementing the Web-STAR IDE, we chose to use technologies that are mature, do not require license fees, have a large community of contributors, and can be deployed easily. Furthermore, all technologies used are available as free and opensource software, and their communities release frequent updates and new capabilities in each new release. These considerations are important for a project that seeks to be expandable, scalable, and easy to maintain.

The system is based on PHP for backend operations, on the MariaDB database for the data storage, and on the JQuery JavaScript library for the front-end design. Behind this infrastructure lies the STAR system (Diakidoy *et al.* 2015) and the SWI-Prolog (Wielemaker *et al.* 2012) 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.

https://doi.org/10.1017/S1471068418000443 Published online by Cambridge University Press

All data storage is handled with 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 interface design, the Bootstrap framework is used. Bootstrap is an HTML, CSS, and JS framework for developing responsive projects on the web. This framework has a number of ready-to-use components like buttons, panels, toolbars, so on, and is also supported by a large community that develops extra components. The JQuery library (`https://jquery.com/`) is 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/`). TogetherJS is a JavaScript library from Mozilla that uses the Web RTC (Johnston and Burnett 2012) technology to enhance communication. It provides audio and chat capabilities between users, and allows users to see each other's mouse cursors and clicks, and the screen content.

The source code editor is based on the ACE editor (`https://ace.c9.io`), an open source web editor which is used by many other popular cloud IDEs. This editor was chosen because of its maturity, its open source license, and for its popularity. 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.

For the visualization of the background knowledge, we sought a component that is able to represent rules in a graph format and can additionally allow interaction with the user and the graph elements. For that reason, Cytoscape.js (Franz *et al.* 2016) was selected, which is an open-source JavaScript-based graph library that allows users to interact with the graph, while supporting both desktop browsers and mobile browsers. It can also handle user events on graph elements like clicking, tapping, dragging, so on. This library also provides a large number of extensions that are employed to enhance the functionality of the Web-STAR IDE. The code folding/unfolding capability uses the "expand-collapse" extension, which provides an API for expanding and collapsing compound parent nodes on a cytoscape graph. The "edge drawing" tool of the visual editor uses the "edgehandles" extension, which provides a user interface for dynamically connecting nodes with edges. The graph navigator capability is based on the "navigator" extension, which provides a bird's eye view (see Figure 12) with pan and zoom control from the graph.

For converting a story from natural language to the STAR syntax, we use a custom-built component developed at our lab, which uses the Stanford CoreNLP for NLP, a python script for processing the NLP output, and PHP for orchestrating and delivering the results through a RESTful API. The Web-STAR IDE integrates this component into its workflow, while the same methodology can be used by other systems to acquire this functionality.

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 in order to process, and for retrieving the results after the completion of the reasoning process (see Figure 5): the "`add_story_queue`" web service takes as a parameter the story in the STAR syntax and returns a unique identifier; the "`retrieve_story_results`" web service takes as a parameter the unique identifier previously sent by the "`add_story_queue`" web service, and

https://doi.org/10.1017/S1471068418000443 Published online by Cambridge University Press

returns the results of the comprehension process. This approach was chosen to minimize the waiting time in cases of large story files that require extensive processing.

5 Web-STAR IDE evaluation

An important step in the design and deployment of a web-based IDE is the evaluation of its usability, that is, "the degree to which users are able to use the system with the skills, knowledge, stereotypes, and experience they can bring to bear" (Eason 2005). Usability evaluation can be conducted using interviews, task analysis, direct observation, questionnaires, and heuristic evaluation, among others (Barnum 2001). In terms of evaluating an IDE, Kline and Seffah (2005) presented three techniques which can also be applied for the Web-STAR IDE's evaluation: (i) the unstructured interviews, (ii) the heuristic evaluation and psychometric assessment, and (iii) the laboratory observation combined with the cognitive walkthrough. Moreover, in work by Pansanato *et al.* (2015), the capturing of user interaction is stretched for usability evaluation of rich web interfaces. The authors present a number of tools and methods that go beyond simple capturing of log files from the web server, like the recording of user interaction from the client side, that is, the browser.

5.1 Evaluation setting

We followed a hybrid approach for the Web-STAR IDE's evaluation that combines the cognitive walkthrough method (John and Packer 1995; Blackmon *et al.* 2002) with questionnaires and user interaction capturing techniques. The process was divided into the design phase, the pilot phase, and the actual evaluation phase, and sought to:

- evaluate the web-interface in terms of ease of use, understanding, learnability, and efficiency;
- detect possible usability problems of the Web-STAR IDE; and
- perform the above for both experts and non-experts that use the IDE.

5.1.1 Design phase

The design phase involved the selection of the participants for the evaluation, the design of the tasks that each participant would undertake, the preparation of the questionnaires, and the technical methods for tracking each participant's interaction with the system.

Participants were chosen from both groups that would have an interest in using the Web-STAR IDE: (i) experts and (ii) non-experts. The expert group included computer scientists and psychologists with prior experience in using the STAR system as a standalone Prolog application, and computer scientists or computer science students with programming skills in Prolog or other declarative programming languages. The non-expert group included psychologists, school teachers, law students, and students of psychology, who had very little or no experience in using IDEs or programming languages. A total of 15 participants were selected, which, according to the relevant bibliography (Macefield 2009), is an appropriate sample for detecting the majority of the usability problems of a system.

https://doi.org/10.1017/S1471068418000443 Published online by Cambridge University Press

We compiled a list of **Cognitive Walkthrough Tasks** that were specifically designed to evaluate the major functions and aspects of the Web-STAR IDE. Each task instructed the user to perform a sequence of actions, as follows:

- **Task 1 (Create an account)**: Navigate to the Web-STAR IDE link and create an account. Activate the account and log into the system.
- **Task 2 (Follow the guided tour)**: Follow the guided tour to learn the basic functionality of the IDE.
- **Task 3a (Write a new story in natural language)**: Write a given story along with its questions in natural language and convert it to the STAR syntax. Then add the background knowledge given in a visual format, using the visual editor and convert it to the STAR syntax. Save the story.
- **Task 3b (Write a new story in the STAR syntax)**: Write a given story along with its questions in symbolic format, using the source code editor. Then add the background knowledge given in the STAR syntax and convert it to the visual format. Save the story.
- **Task 4 (Load a story and initiate the comprehension process)**: Choose a public story, load it, and initiate the story comprehension process.
- **Task 5a (Modify the background knowledge using the visual format editor)**: Load a story, add a new background knowledge rule given in visual format, update and remove an existing rule, all using the visual editor. Finally, initiate the comprehension process.
- **Task 5b (Modify the background knowledge using the source code editor)**: Load a story, add a new background knowledge rule given in the STAR syntax, update and remove an existing rule, all using the source code editor. Finally, initiate the story comprehension process.
- **Task 6 (Filter the output of the comprehension process)**: Load a story, initiate the comprehension process, and filter the output to present only the concepts that change while the story unfolds.
- **Task 7 (Share a story)**: Load a story and share it in the public story repository.
- **Task 8 (Comment on a user's story)**: Find a story in the public story repository and add a comment on that story.
- **Task 9 (Initiate the collaboration tool)**: Initiate the collaboration functionality and send the generated collaboration link to another person using the feedback option.

Since not all participants are experts in encoding stories in a symbolic language, one of the major considerations when preparing these tasks was to obtain comparable results from the users. Toward that aim, both the story and the background knowledge in Tasks 3a and 3b were provided in the instructions given to the users.

After observing each participant performing the above tasks, the experimenters tried to answer the following **Cognitive Walkthrough Questions**, as explained in the work of Wharton *et al.* (1994):

- Does the user try to achieve the right effect?
- Does the user notice that the correct action is available?

https://doi.org/10.1017/S1471068418000443 Published online by Cambridge University Press

- Does the user associate the correct action with the effect that the user is trying to achieve?
- If the correct action is performed, does the user see that progress is being made toward the solution of the task?

When the answer to any of these questions was "No," an error was counted toward the total number of errors for that task.

The time needed to complete each task was calculated from the time the participant logged into the IDE until the time the participant logged out of it, with an exception in the first task where the time was measured from the time the participant clicked the register button until the time the participant logged out of the IDE.

After the completion of the tasks, a **Demographics Questionnaire** was completed by participants to record their gender, age, degree, occupation, previous experience in using IDEs, knowledge of programming languages, and prior experience in using story understanding systems and more specifically the STAR system. A **Post-task Questionnaire** was also completed to capture the participants' opinion for using the IDE for each specific task. The questionnaire included questions that covered the various parts of the system invoked for each task: the interface (e.g. menu bar, panels, dialogs, buttons, and labels), the online help material, the visual editor, the public story repository, and the outcome of the story comprehension process. It included true/false questions, multiple choice questions, and questions in the five-point Likert scale. The questionnaire is shown in Appendix B (supplementary material). The questionnaire also included a section with questions from the **System Usability Scale (SUS) standardized questionnaire** (Brooke 1996), a 10-item questionnaire using a five-point scale for the assessment of perceived usability (Lewis *et al.* 2015). Both surveys were designed and deployed online and access to them was restricted to participants of our evaluation.

Finally, we implemented a logging functionality to capture detailed information from the participants' interaction with the Web-STAR IDE (e.g. login, menu selection, button click, and visual editor usage) and measure the time between these interactions. This functionality was seamlessly integrated with the Web-STAR IDE using AJAX technology. Each event was stored in a database table and included the user-id of the participant that performed the action, the time the action was performed, the component used (e.g. login screen, menu bar, and visual editor), the action (e.g. button click, visual editor graph node added), the data sent, and the response of the IDE.

The metrics chosen for the evaluation were both qualitative (e.g. user satisfaction, ease of use) and quantitative (e.g. number of successfully performed tasks, task completion time, number of errors occurred, number of times participants used the online help functionality, number of times participants clicked on a control).

5.1.2 Pilot phase

Before the actual evaluation phase, we performed a pilot evaluation identical to the actual one, but with only two users, to verify that all tasks are feasible and understandable. This also allowed us to test that data were recorded properly and to get familiar with the testing process.

The pilot evaluation was performed in a laboratory environment with a computer connected to the Internet with access to the Web-STAR IDE URL. We also set up the

https://doi.org/10.1017/S1471068418000443 Published online by Cambridge University Press

screen recorder software to capture all interactions of the user with the interface (e.g. keystrokes, mouse movements, information dialogs, and visual editors) in a video file.

Both participants ended up needing more than an hour to complete the tasks and respond to the questionnaires.

5.1.3 Evaluation phase

All participants in the evaluation phase performed the experiment in a controlled environment which included a laptop with an Intel® CORE™ i7 processor, 4GB of RAM, and a 15.4 inches screen. An external mouse was attached to the laptop, and participants had instructions to use it (instead of the laptop's integrated mousepad). The laptop was constantly plugged into the power source. In terms of software, the Firefox web-browser was used to load the WebSTAR IDE interface, and the Camtasia screen recording software was activated before each session to record the participant's actions. Each session was performed in a quiet room with only the participant and the experimenter present, aiming to minimize outside interference and noise from the environment.

As a first step, each participant completed a statement of informed consent regarding the reason for the evaluation, and the data collection and data handling policy. This consent was mandatory for participating in the evaluation. The participants were then asked to complete the Demographics Questionnaire online.

Following that, participants were presented with a document listing the Cognitive Walkthrough Tasks. The document is presented in Appendix A (supplementary material). During the cognitive walkthrough, participants had continuous access, through the Web-STAR IDE, to online help files provided by the IDE, the STAR syntax guide, and the guided tour; that is, the same type of help that any typical user of the IDE would have available while using it. Participants had the option to choose any type of viewing mode they saw fit when completing the tasks.

During each task, the experimenter recorded all observations made in a notebook, answered the cognitive walkthrough questions, and recorded problems and errors occurred while the participant used the IDE. After each task, the participant was presented with the post-task questionnaire for that specific task. The experimenter avoided providing any kind of verbal or non-verbal additional help to the participant while conducting the cognitive walkthrough.

After the completion of all the tasks, participants were presented with the SUS standardized questionnaire. Finally, the experimenter stopped the screen recording, stored the capture, and saved all questionnaire answers online for later processing.

5.2 Evaluation results

Fifteen people (eight male, seven female) participated voluntarily to the evaluation. All had Greek as their mother tongue and reported to have a normal or corrected-to-normal vision. All participants completed the whole evaluation process. Figure 15 represents analytics regarding their gender, age group, education, employment status, and knowledge of programming languages and IDEs. More than half of the participants (8 out of 15) reported that they had heard the notion of story understanding, but only two reported

https://doi.org/10.1017/S1471068418000443 Published online by Cambridge University Press

Table 1. *Performance at the cognitive walkthrough evaluation*

Task	Completed[a]	Avg[b]	Std[c]	Max[d]	Min[e]	Errors[f]
Task 1	100%	111	45	202	37	0
Task 2	100%	692	269	1303	224	0
Task 3a	100%	640	168	1016	387	0
Task 3b	100%	119	23	156	74	0
Task 4	100%	186	95	381	100	0
Task 5a	100%	615	105	861	438	0
Task 5b	100%	244	58	338	164	0
Task 6	100%	85	19	114	54	0
Task 7	100%	98	40	186	41	0
Task 8	100%	107	42	230	56	0
Task 9	100%	75	23	132	38	0

[a] Percentage of participants that successfully completed the task.
[b] Average time (in seconds) needed to complete the task.
[c] Standard time deviation (in seconds) needed to complete the task.
[d] Maximum time (in seconds) needed to complete the task.
[e] Minimum time (in seconds) needed to complete the task.
[f] Number of errors recorded by the experimenter during the task.

that they had used a story understanding system before. In both cases, this system was the STAR system. The group of non-experts included 10 participants and the group of experts 5.

After the completion of the evaluation, the notes taken by the examiner for each participant with answers to the cognitive walkthrough questions were carefully examined along with the answers in the post-evaluation questionnaire. The log files of each participant were also analyzed, and the aggregated results are presented in Table 1 and Figure 16.

The average total time for completing the evaluation tasks from both groups satisfy the normality assumption based on the Kolmogorov–Smirnov test. On average, experts performed the tasks of the cognitive walkthrough in less time (M=2622.40 seconds, SE=107.44) than non-experts (M = 3150.00 seconds, SE = 98.01), and the difference was significant $t(13) = 3.32$, $p < .05$.

Further analysis of the results per task gives more insights on how participants interacted with the system (see Table 1).

5.2.1 Results from the cognitive walkthrough process

In the following paragraphs we present the findings of the cognitive walkthrough process:

Task 1: All participants completed this task successfully. Some participants did not receive the activation email immediately due to email provider delays. The majority of participants clicked the links included in the introductory text while some others chose the "Register" option from the menu bar. It was also common for participants to try and press the enter button after filling up their credentials, but it was not working and they needed to click the "Login" button instead to proceed.

https://doi.org/10.1017/S1471068418000443 Published online by Cambridge University Press

https://doi.org/10.1017/S1471068418000443 Published online by Cambridge University Press

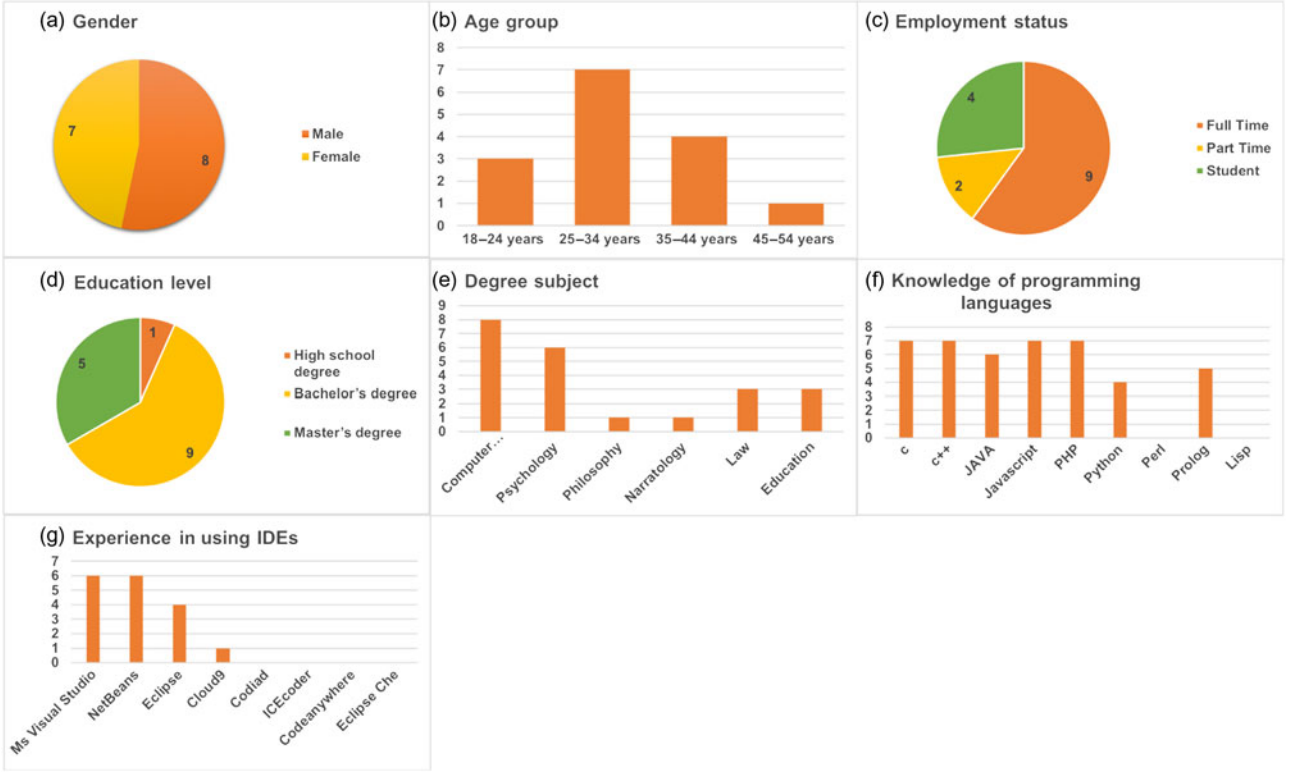


Fig. 15. Participants' demographics. Graph A depicts their gender, Graph B their age group distribution, Graph C their employment status, Graph D their education level, Graph E their degree subject, Graph F their knowledge of programming languages, and Graph G their experience in using IDEs.

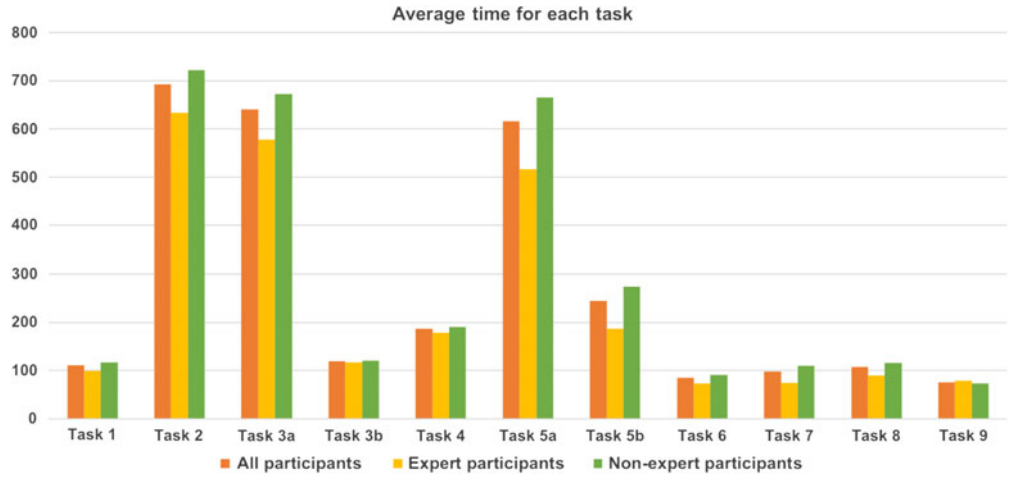


Fig. 16. Average time in seconds per task, for experts, non-experts, and all participants.

Task 2: All participants completed this task successfully. The majority of participants initiated the guided tour using the assistant dialog. A small number of participants did not understand that the assistant's message is clickable, and used the "Help menu" to find and start the guided tour. Some participants interacted with the IDE while following the guided tour and performed actions like "reading a story", "drawing background knowledge" using the visual editor and converting both the story and the background knowledge to test it. There were cases of participants who went back to a specific step, to test a functionality mentioned later in the guided tour. Moreover, one participant also expressed the opinion that it would be very useful if there was an option to watch a video instead of the guided tour. When the guided tour was showcasing the output panel, participants expected to have the story comprehension output area filled up with story information, but it was empty, since the story comprehension process was not activated by the guided tour.

Task 3a: All participants completed this task successfully. Some participants watched the help video first to properly perform the task. Participants used the visualizations, that is, highlight of literals and rules which can be connected with an edge while drawing it, and red highlighting for incomplete rules along with debugging messages. Some participants were double clicking on the nodes and edges to move them and add arguments, when only a single click was needed.

Task 3b: All participants completed this task successfully. Participants found easily where to add the story and the background knowledge. A number of participants who tested the toolbar of the source code editor used functions and controls like the "text wrap" and the "font size."

Task 4: All participants completed this task successfully. Some participants had difficulties finding the "Read Story" button and they tried to locate it on the menu bar or on the top area of the IDE. Furthermore, non-experts read the confirmation message to understand where they could find the story output and when the reading process was completed.

Task 5a: All participants completed this task successfully. Some non-experts deleted the rule, but they forgot to delete the connected literal, whereas experts deleted the

https://doi.org/10.1017/S1471068418000443 Published online by Cambridge University Press

connected literal along with the rule. When the former users tried to convert the graph to the STAR syntax, the system's debugging messages guided them to delete the connected literal as well before proceeding with the conversion. Some participants did not notice that some literals were existing and when they tried to add them, the debugging messages informed them that the literal they were trying to add already existed, and then they proceeded with connecting the existing literal with the rule.

Task 5b: All participants completed this task successfully. The majority of experts, when instructed to delete a rule, they commented it out, whereas non-experts proceeded with erasing it. Some experts also used the search functionality to find the rule and then delete it.

Task 6: All participants completed this task successfully. They found the filtering options very easily. Some participants tried to use the filtering option before the story comprehension process was completed and they could not, since the option was available only after the completion of the process. Hence, they waited for the reading process to finish and then tried to apply the filter.

Task 7: All participants completed this task successfully. The majority of participants had difficulty in locating the share button. First, they searched for it on the menu bar and then at the story area. Only after careful examination of the screen they were able to locate it. Some participants browsed to the save story window and chose the "private/public" toggle to share the story. In most cases, participants scrolled up and down the IDE page to find the relevant control to share a story.

Task 8: All participants completed this task successfully. Some participants had difficulties locating how to comment on a story. They searched for the button on the menu bar and then they navigated to the public story repository to find the commenting functionality.

Task 9: All participants completed this task successfully. Some participants did not locate the "Start Collaboration" button immediately and searched for it in the public story repository.

5.2.2 Results from the post-task questionnaire

Results from the post-task questionnaire show that for **Task 1**, on average, participants strongly agree that the process of creating a new account (M_E=5.0, M_{NE}=4.9)[2] and activating it (M_E=4.8, M_{NE}=4.9) is easy and is the same (for creating, M_E=4.6, M_{NE}=4.9), (for activation, M_E=4.8, M_{NE}=5.0) with that of the other systems they are using.

For **Task 2**, on average, experts agree and non-experts strongly agree that it is easy to find and start the guided tour (M_E=4.4, M_{NE}=4.8). On average, both experts and non-experts strongly agree that the duration of the guided tour is appropriate for learning the basics of the IDE (M_E=4.6, M_{NE}=4.5). In terms of feeling confident in using the IDE after the guided tour, on average, experts strongly agree that this is the case and non-experts agree as well (M_E=4.6, M_{NE}=3.9).

For **Task 3a**, on average, participants strongly agree that it is easy to write the story in natural language (M_E=5.0, M_{NE}=4.9) and automatically convert it to the STAR

[2] M_E and M_{NE} represent the means of the Likert scale scores given by expert and non-expert participants, respectively.

https://doi.org/10.1017/S1471068418000443 Published online by Cambridge University Press

syntax (M_E=5.0, M_{NE}=5.0). On average, experts agree and non-experts strongly agree that it is easy to add the background knowledge of the story using the visual editor (M_E=4.4, M_{NE}=4.5). Both groups strongly agree that the automatic conversion of the background knowledge in visual format to the STAR syntax is easy (M_E=5.0, M_{NE}=5.0). As for saving the story, participants strongly agree that it is an easy task (M_E=5.0, M_{NE}=5.0). Four non-experts have used the online help facility to perform this task and on average, they strongly agree that the help available from the system to perform this task is adequate (M_{NE}=4.8).

For **Task 3b**, on average, participants strongly agree that it is easy to write the story in the STAR syntax (M_E=5.0, M_{NE}=4.6). On average, experts agree and non-experts strongly agree that it is more efficient to write the story in natural language and then convert it to the STAR syntax than writing the story directly using the STAR syntax (M_E=4.4, M_{NE}=5.0). On average, participants strongly agree that it is easy to add the background knowledge in the source code editor (M_E=5.0, M_{NE}=4.9). One non-expert stated that this does not apply. Both groups on average, strongly agree that it is easy to convert the background knowledge from the STAR syntax to visual format (M_E=5.0, M_{NE}=5.0). Both experts and non-experts, on average, agree that it is easier to understand the background knowledge rules in visual format than in the STAR syntax (M_E=3.6, M_{NE}=4.3). Although the means of the two groups on this question appear to have a difference larger than that of other questions, further analysis revealed that this difference was not found to be statistically significant based on the Mann–Whitney test, U=14.00, z= −1.42, p>.05, r= −0.37. Participants strongly agree that it is easy to save the story. None of the participants has used the online help facility to perform this task.

For **Task 4**, on average, participants strongly agree that it is easy to find a story and load it (M_E=5.0, M_{NE}=5.0) and that the story load window is easy to use (M_E=5.0, M_{NE}=4.9). On average, experts agree and non-experts strongly agree that it is easy to find how to initiate the story comprehension process (M_E=4.2, M_{NE}=4.8). On average, both groups strongly agree that the system provides continuous feedback on the comprehension process status (M_E=4.8, M_{NE}=4.9). In terms of finding the answer that the system gave to a question, on average, both experts and non-experts strongly agree that it is easy to find the answer to the question using the visual output panel (M_E=5.0, M_{NE}=4.7). One expert participant stated that this does not apply since he/she used only the raw output. On average, experts strongly agree and non-experts agree that it is easy to find the answer to the question using the raw output panel (M_E=5.0, M_{NE}=4.0). Five participants (one expert and four non-experts) stated that this does not apply since they used only the visual output. On average, both experts and non-experts strongly agree that the visual output panel presents the story concepts and questions in an understandable way (M_E=4.8, M_{NE}=4.7). For the raw output panel, on average, experts strongly agree and non-experts agree that it presents the various story concepts and questions in an understandable way (M_E=5.0, M_{NE}=4.0). Five participants (one expert and four non-experts) stated that this does not apply since they used only the visual output. Two participants (one from each group) used the online help facility and all participants were able to find the correct answer to the question.

For **Task 5a**, on average, experts agree and non-experts strongly agree that it is easy to add a rule using the background knowledge visual editor (M_E=4.4, M_{NE}=4.9).

https://doi.org/10.1017/S1471068418000443 Published online by Cambridge University Press

On average, both experts and non-experts strongly agree that it is easy to delete (M_E=4.6, M_{NE}=4.8) and edit (M_E=5.0, M_{NE}=4.8) a rule using the background knowledge visual editor. Regarding the controls available in the background knowledge visual editor, on average, experts strongly agree and non-experts agree that they are easy to use (M_E=4.6, M_{NE}=4.4). On average, both experts and non-experts strongly agree that it is easy to understand the functionality of the controls in the visual editor's toolbar (M_E=4.8, M_{NE}=4.6). Only one non-expert participant has used the online help facility. All participants were able to find the correct answer to the question.

For **Task 5b**, on average, both experts and non-experts strongly agree that it is easy to add, delete and edit a rule using the background knowledge source code editor (M_E=5.0, M_{NE}=5.0). Moreover, on average, both groups strongly agree that the controls available in the background knowledge source code editor's toolbar are easy to use (M_E=4.8, M_{NE}=5.0). On average, experts strongly agree and non-experts agree that it is easy to understand what is the functionality of the controls in the background knowledge source code editor (M_E=4.8, M_{NE}=4.9). One non-expert participant stated that this does not apply. In terms of what is the most efficient method to modify the background knowledge, on average, experts neither agree nor disagree that it is the visual editor, whereas non-experts agree that the visual editor is more efficient than the source code editor (M_E=2.8, M_{NE}=4.0). One non-expert participant stated that this does not apply. This difference between the means of the two groups was not found to be statistically significant based on the Mann–Whitney test, U=10.50, z= −1.65, p>.05, r= −0.44. None of the participants had used the online help facility. All participants but one, were able to find the correct answer to the question.

For **Task 6**, on average, both experts and non-experts strongly agree that it is easy to find and apply the filtering functionality (M_E=5.0, M_{NE}=5.0). Moreover, on average, participants strongly agree that the filters available can help extract information from the comprehension model (M_E=4.8, M_{NE}=4.9). None of the participants had used the online help facility.

For **Task 7**, on average, both experts and non-experts strongly agree that it is easy to find a demo story and load it (M_E=5.0, M_{NE}=5.0) and that the story browser window is easy to use (M_E=5.0, M_{NE}=5.0). On average, both groups agree that it is easy to find how to share a story (M_E=4.0, M_{NE}=3.5). None of the participants had used the online help facility.

For **Task 8**, on average, experts agree and non-experts strongly agree that it is easy to find a story in the public story repository (M_E=4.4, M_{NE}=4.9). On average, both groups agree that it is easy to comment on a story (M_E=4.4, M_{NE}=4.4) and strongly agree that comments added by others are clearly presented on the screen (M_E=4.8, M_{NE}=4.6).

For **Task 9**, on average, both experts and non-experts strongly agree that it is easy to find how to initiate the collaboration functionality (M_E=4.6, M_{NE}=4.9). On average, experts agree and non-experts strongly agree that the collaboration functionality could be useful for teaching logic programming (M_E=4.4, M_{NE}=4.7), collaboratively creating stories (M_E=4.4, M_{NE}=4.7) and collaboratively designing knowledge (M_E=4.2, M_{NE}=4.7).

For all tasks, on average, participants strongly agree that the feedback messages from the system are helpful.

https://doi.org/10.1017/S1471068418000443 Published online by Cambridge University Press

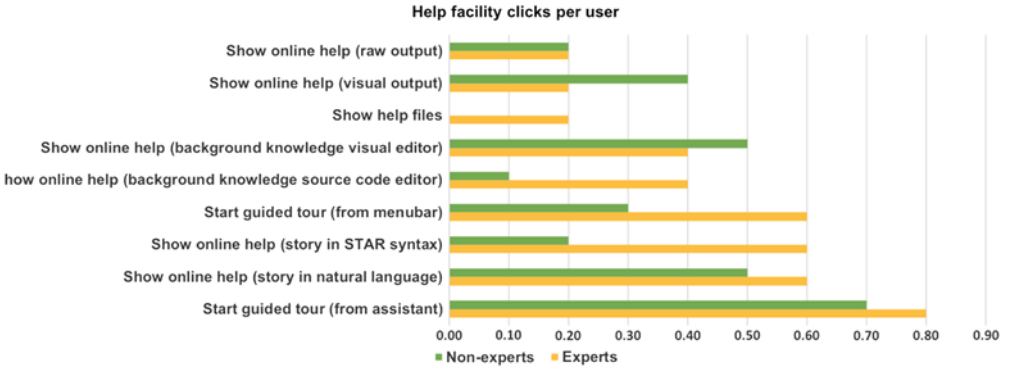


Fig. 17. Mean number of clicks per user for expert and non-expert participants on the help options of the IDE.

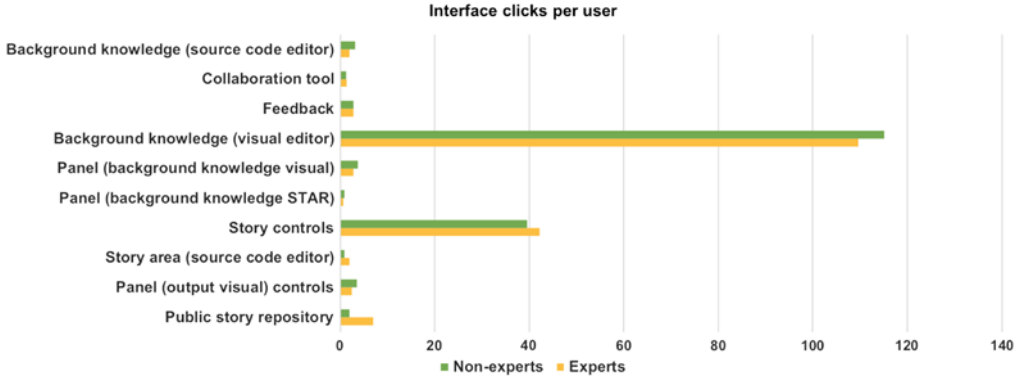


Fig. 18. The 10 most clickable parts of the interface per participant. The X-axis represents the mean number of clicks.

5.2.3 Results from the logging functionality

During the experiment, all participants' interactions with the IDE were captured and stored in the database. The clicks per user for both experts and non-experts are presented in the following graphs, with a focus on the clicks on the help facilities (see Figure 17), and on the 10 most clicked functions per user (see Figure 18).

As the results show, the background knowledge visual editor is the most clickable area. This was expected since participants had to draw and edit knowledge rules using the visual editor. In general, experts and non-experts had little difference in the number of clicks per area and function.

5.2.4 Results from the SUS questionnaire

Results from the SUS standardized questionnaire show an average score of **88.33** out of **100**. The maximum score of the participants was 100, the minimum was 70 and the standard deviation was 8.5. Results are depicted in Table 2 for both groups as well as for the entire set of participants.

https://doi.org/10.1017/S1471068418000443 Published online by Cambridge University Press

Table 2. *Results of the SUS standardized questionnaire*

Group	Average score	Std[a]	Max[b]	Min[c]
Experts	90.00	7.07	95.00	77.50
Non-experts	88.25	9.43	100	70
TOTAL	**88.83**	8.50	100	70

[a] Standard deviation.
[b] The minimum score.
[c] The maximum score.

Compared to the SUS scores obtained from the evaluations of other systems, the Web-STAR is ranked in the top category, that is, between "excellent" and "best imaginable" in the adjective ratings scale (Bangor *et al.* 2009).

5.3 Analysis of results

The evaluation process followed allowed a thorough investigation of the participants' actions, impressions, and feedback while using the IDE. The combination of a cognitive walkthrough, with questionnaires, and with close monitoring offered information that could not have been obtained if using only a single method for evaluation. The diverse group of participants in this evaluation gives insights into how people from different backgrounds and prior experience in using story understanding systems and IDEs in general can benefit from the various features of the Web-STAR IDE.

After examination of the results, we report that all participants, experts and non-experts, managed to complete all the tasks. In general, participants did not have much difficulty while performing the tasks. For some tasks, like sharing a story and commenting on it, participants had some trouble finding the relevant controls since they were not in the "expected" area of the IDE (e.g. the menu bar).

Both experts and non-experts managed to setup an account, activate it and access the IDE in less than 2-min time. Participants were able to do that because the registration process is similar to that of other online systems they already have accounts on and use. They were able to start using the IDE in a very short time, by following the guided tour.

For the main task that the IDE facilitates, which is writing stories, both experts and non-experts were able to encode stories either by converting them from natural language to the STAR syntax or by writing them directly in the STAR syntax. Regarding the background knowledge, participants were able to encode it easily using the visual editor and the source code editor (even if they just had to copy the prepared story in symbolic format). All participants agree that it is easier to write the story in natural language and then convert it to the STAR syntax than writing it directly in symbolic format. Moreover, all participants were able to understand the background knowledge rules when using the visual editor and the graph representation of the background knowledge. This was clear by the answers given to the post-task questionnaire and from the time the participants took to complete the relevant tasks. This is important, since participants can use the component that best fits their working style and needs, to perform this action.

https://doi.org/10.1017/S1471068418000443 Published online by Cambridge University Press

For editing the background knowledge, expert participants found the usage of the source code editor more efficient than that of the visual editor. This is to be expected, since it is presumably more time-consuming to draw a rule using the visual editor than to write it in the source code editor. For non-expert users, this was clearly not the case, since they agree that the visual editor is more efficient for changing the background knowledge. We assume that this could be because they can understand better the graph representation of the knowledge instead of the STAR syntax that they are not familiar with.

In terms of finding the answer to the questions posed, all participants were able to perform this task quite easily using either the visual or the raw output panel. Experts preferred the raw output which was enhanced with color highlighting for questions and answers and non-experts preferred the visual output with the time-line format. A number of non-expert participants chose to use only the visual output to find the answer. This is justified by the fact that the answer could be easily extracted from the time-line without the need to explore the raw output.

At this point, we observed that when a story had several scenes and a participant tried to find the answer to a question from the first scene using the raw output, he/she needed to scroll up to find it. Hence, we decided to add the option in the next version of the system to split the raw output to scenes, so that this user burden can be avoided, by making it easier to browse each scene from the raw output. Both groups benefited from the time-line format since it was easier to understand the various concepts of each story, apply filters on them, and find answers to questions.

The social and collaboration features of the IDE were also simple to use. Both groups were able to share a story or add comments to a story in the public story repository in a very short time. We observed that a number of participants had a problem spotting the relevant controls, since they were not located in the expected area. Hence, we decided to add a menu option that groups all these controls and buttons together for easy access in the next version of the system. Participants also found the collaboration feature very useful, since they confirmed that it could be useful for teaching logic programming, collaboratively creating stories, and collaboratively designing knowledge.

Results from the SUS standardized questionnaire dictate that the Web-STAR IDE is a friendly, easy to use, and easy to learn IDE. This evaluation led to some minor changes in the IDE to enhance user experience and productivity.

6 Conclusion and future work

We have presented Web-STAR, a platform built to facilitate the interaction of users with the STAR system for story comprehension. We have discussed the various features of the platform through examples, and have argued that the platform is designed to appeal to both expert and non-expert users. The argument is supported, in particular, by the visualization that Web-STAR offers for the background knowledge that is used during story comprehension, and for the output of the story comprehension process. A comprehensive evaluation of the usability of the platform has supported that the platform is, indeed, friendly, easy to use, and easy to learn.

The platform is currently used for educational purposes, helping students and researchers engage with the problem of automated story understanding. More than 40 users have registered so far, and have contributed more than 70 stories. Moreover, the

https://doi.org/10.1017/S1471068418000443 Published online by Cambridge University Press

platform has received more than 1350 web service calls for processing STAR domain files. The platform was also used in the Robot Trainer Game (Rodosthenous and Michael 2016), a crowdsourcing game with a purpose to gather commonsense knowledge; in this context, knowledge contributed by users was processed in real time to determine its sufficiency to answer story questions. The demonstrable ease of use of the Web-STAR platform, and its online and visual environment, makes it a prime candidate for use by domain experts in law, history, or literature, who may wish to comprehend text in the form of narratives.

Future versions of the platform will aim to refine its interface and extend its functionality. In terms of the latter, we are considering the addition of the option to import and process relevant background knowledge from existing knowledge bases like Conceptnet (Speer and Havasi 2013), YAGO (Mahdisoltani *et al.* 2015), NELL (Mitchell *et al.* 2015), and the OpenCyc project (Lenat 1995); or directly from Games With A purpose (von Ahn and Dabbish 2008) or other crowdsourcing platforms like Amazon Mechanical Turk (Buhrmester *et al.* 2011); or even from machine learning algorithms that produce rule-based knowledge bases (Michael 2009; Michael 2016; Michael 2017). The component that converts natural language stories into the STAR syntax could be further extended by incorporating systems that extract knowledge from natural language (Corcoglioniti *et al.* 2016) and identify the temporal ordering of events (UzZaman *et al.* 2013).

It is our hope that this work will serve as a basis for establishing a story-sharing and story-processing community, toward the advancement of work in automated story understanding through symbolic knowledge and reasoning.

Supplementary material

To view supplementary material for this article, please visit https://doi.org/10.1017/S1471068418000443.

References

Abbott, H. P. 2008. *The Cambridge Introduction to Narrative*. Cambridge University Press.

Balai, E., Gelfond, M. and Zhang, Y. 2013. *Towards Answer Set Programming with Sorts*. Springer, Berlin, Heidelberg, 135–147.

Bangor, A., Kortum, P. and Miller, J. 2009. Determining what individual SUS scores mean: Adding an adjective rating scale. *Journal of Usability Studies 4,* 3, 114–123.

Barnum, C. M. 2001. *Usability Testing and Research*, 1st ed. Allyn & Bacon, Inc., Needham Heights, MA, USA.

Baroni, P., Caminada, M. and Giacomin, M. 2011. An introduction to argumentation semantics. *The Knowledge Engineering Review 26,* 4, 365–410.

Bench-Capon, T. and Dunne, P. E. 2007. Argumentation in artificial intelligence. *Artificial Intelligence 171,* 10–15, 619–641.

Besnard, P. and Hunter, A. 2008. *Elements of Argumentation*, vol. 47. MIT Press, Cambridge, MA, USA.

Blackmon, M. H., Polson, P. G., Kitajima, M. and Lewis, C. 2002. Cognitive walkthrough for the web. In *Proc. of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '02. Minneapolis, Minnesota, USA, 20–25 Apr. 2002, ACM, New York, NY, USA, 463–470.

https://doi.org/10.1017/S1471068418000443 Published online by Cambridge University Press

Bogaard, T., Wielemaker, J., Hollink, L. and Van Ossenbruggen, J. 2017. *SWISH Data-Lab: A Web Interface for Data Exploration and Analysis*. Springer International Publishing, Cham, Switzerland, 181–187.

Brandes, U., Eiglsperger, M., Lerner, J. and Pich, C. 2013. Graph Markup Language (GraphML). In *Handbook of Graph Drawing Visualization*, R. Tamassia, Ed. Discrete mathematics and its applications. CRC Press, Boca Raton, FL [u.a.], 517–541.

Brooke, J. 1996. SUS – A quick and dirty usability scale. *Usability Evaluation in Industry 189,* 194, 4–7.

Buhrmester, M., Kwang, T. and Gosling, S. D. 2011. Amazon's Mechanical Turk. *Perspectives on Psychological Science 6,* 1, 3–5.

Carroll, N. 2001. On the narrative connection. In *Beyond Aesthetics: Philosophical Essays*, Cambridge University Press, 118–133.

Charniak, E. 1972. *Toward a Model of Children's Story Comprehension*. Tech. Rep. AITR-266, Cambridge, MA, USA.

Charniak, E. 1977. Ms. Maloprop, a language comprehension program. In *Proc. of the 5th International Joint Conference on Artificial Intelligence*, vol. 1. IJCAI'77, Cambridge, Massachusetts, USA, 22–25 Aug. 1977, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1–7.

Cloud9 IDE. 2017. Cloud9 IDE. URL: http://c9.io/. [Accessed on April 18, 2017].

Codeanywhere Inc. 2017. Codeanywhere IDE. URL: https://codeanywhere.com/. [Accessed on April 18, 2017].

Corcoglioniti, F., Rospocher, M. and Palmero Aprosio, A. 2016. Frame-based ontology population with PIKES. *IEEE Transactions on Knowledge and Data Engineering 28,* 12, 3261–3275.

Dasseville, I. and Janssens, G. 2015. A web-based IDE for IDP. In *1st International Workshop on User-Oriented Logic Programming (IULP2015)*.

Davis, E. 1990. *Representations of Commonsense Knowledge*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Denecker, M. and Ternovska, E. 2008. A logic of nonmonotone inductive definitions. *ACM Transactions on Computational Logic 9,* 2 (Apr.), 14:1–14:52.

Diakidoy, I.-A., Kakas, A., Michael, L. and Miller, R. 2014. Story comprehension through argumentation. In *Proc. of the 5th International Conference on Computational Models of Argument (COMMA 2014)*, Scottish Highlands, UK, 9–12 Sep. 2014, S. Parsons, N. Oren, C. Reed and F. Cerutti, Eds. IOS Press, 31–42.

Diakidoy, I.-A., Kakas, A., Michael, L. and Miller, R. 2015. 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)*, 64–70.

Eason, K. D. 2005. *Information Technology and Organisational Change*. Taylor & Francis.

Eclipse Foundation. 2017. Eclipse Che IDE. URL: https://eclipse.org/che/. [Accessed on April 18, 2017].

Forster, E. M. 2010. *Aspects of the Novel*. RosettaBooks.

Franz, M., Lopes, C. T., Huck, G., Dong, Y., Sumer, O. and Bader, G. D. 2016. Cytoscape.js: A graph theory library for visualisation and analysis. *Bioinformatics 32,* 2, 309–311.

Friedman, M. 1974. Explanation and scientific understanding. *The Journal of Philosophy 71,* 1, 5–19.

Genette, G. 1982. *Figures of Literary Discourse*. European perspectives: A series in social thought and cultural criticism. Columbia University Press.

https://doi.org/10.1017/S1471068418000443 Published online by Cambridge University Press

Goldman, S. R., Graesser, A. C. and Broek, P. V. D. 1999. *Narrative Comprehension, Causality, and Coherence: Essays in Honor of Tom Trabasso*. Taylor & Francis.

Hirschman, L., Light, M., Breck, E. and Burger, J. D. 1999. Deep read: A reading comprehension system. In *Proc. of the 37th Annual Meeting of the Association for Computational Linguistics on Computational Linguistics*, College Park, Maryland, USA, 20–26 Jun. 1999. Association for Computational Linguistics, Stroudsburg, PA, USA, 325–332.

ICEcoder Ltd. 2017. ICEcoder IDE. URL: https://icecoder.net/. [Accessed on April 18, 2017].

Iyyer, M., Manjunatha, V., Guha, A., Vyas, Y., Boyd-Graber, J. L., Daume, H. III and Davis, L. S. 2016. The amazing mysteries of the gutter: Drawing inferences between panels in comic book narratives. *CoRR abs/1611.0*.

John, B. E. and Packer, H. 1995. Learning and using the cognitive walkthrough method: A case study approach. In *Proc. of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '95. Denver, Colorado, USA, 07–11 May, I. R. Katz, R. Mack, L. Marks, M. B. Rosson and J. Nielsen, Eds. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 429–436.

Johnston, A. B. and Burnett, D. C. 2012. *WebRTC: APIs and RTCWEB Protocols of the HTML5 Real-Time Web*. Digital Codex LLC, USA.

Kakas, A. and Michael, L. 2016. Cognitive systems: Argument and cognition. *IEEE Intelligent Informatics Bulletin 17,* 1, 14–20.

Katz, B. 1997. Annotating the World Wide Web using natural language. In *Computer-Assisted Information Searching on Internet*. RIAO '97. LE CENTRE DE HAUTES ETUDES INTERNATIONALES D'INFORMATIQUE DOCUMENTAIRE, Paris, France, 136–155.

Kline, R. B. and Seffah, A. 2005. Evaluation of integrated software development environments: Challenges and results from three empirical studies. *International Journal of Human-Computer Studies 63,* 6 (Dec.), 607–627.

Kowalski, R. and Sadri, F. 2016. Programming in logic without logic programming. *Theory and Practice of Logic Programming 16,* 3 (May), 269–295.

Lenat, D. 1995. CYC: A large-scale investment in knowledge infrastructure. *Communications of the ACM 38,* 11, 33–38.

Lenat, D. B. and Guha, R. V. 1990. *Building Large Knowledge-Based Systems: Representation and Inference in the Cyc Project*, 1st ed. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Lewis, J. R., Utesch, B. S. and Maher, D. E. 2015. Measuring perceived usability: The SUS, UMUX-LITE, and AltUsability. *International Journal of HumanComputer Interaction 31,* 8, 496–505.

Macefield, R. 2009. How to specify the participant group size for usability studies: A practitioner's guide. *Journal of Usability Studies 5,* 1, 34–45.

Mahdisoltani, F., Biega, J. and Suchanek, F. M. 2015. YAGO3: A knowledge base from multilingual wikipedias. In *Proc. of 7th Biennial Conference on Innovative Data Systems Research (CIDR)*, Asilomar, California, USA, 4–7 Jan. 2015, www.cidrdb.org [Online proceedings], 1–11.

Manning, C. D., Bauer, J., Finkel, J., Bethard, S. J., Surdeanu, M. and McClosky, D. 2014. The Stanford CoreNLP natural language processing toolkit. In *Proc. of the 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, Baltimore, Maryland, USA, 22–27 Jun. 2014, Association for Computational Linguistics (ACL), 55–60.

Marcopoulos, E., Reotutar, C. and Zhang, Y. 2017. An online development environment for answer set programming. In *2nd International Workshop on User-Oriented Logic Paradigms (IULP 2017)*.

https://doi.org/10.1017/S1471068418000443 Published online by Cambridge University Press

Michael, L. 2009. Reading between the lines. In *Proc. of the 21st International Joint Conference on Artificial Intelligence (IJCAI 2009)*, Pasadena, California, USA, 11–17 Jul. 2009, IJCAI Organization, 1525–1530.

Michael, L. 2013. Story understanding... Calculemus. In *Proc. of the 11th International Symposium on Logical Formalizations of Commonsense Reasoning (Commonsense'13)*, Ayia Napa, Cyprus, 27–29 May 2013, [online proceedings].

Michael, L. 2016. Cognitive reasoning and learning mechanisms. In *4th International Workshop on Artificial Intelligence and Cognition*, vol. 1895, New York City, NY, USA, 6–16 Jul. 2016, A. Lieto, M. Bhatt, A. Oltramari and D. Vernon, Eds. CEUR Workshop Proceedings (CEUR-WS.org), 2–23.

Michael, L. 2017. The Advice Taker 2.0. In *Proc. of the 13th International Symposium on Commonsense Reasoning (Commonsense 2017)*, vol. 2052. London, UK, 6–8 Nov. 2017, A. S. Gordon, R. Miller and G. Turán, Eds. CEUR Workshop Proceedings (CEUR-WS.org).

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. and Welling, J. 2015. Never-ending learning. In *AAAI Conference on Artificial Intelligence*, 2302–2310.

Modgil, S. and Prakken, H. 2014. The ASPIC+ framework for structured argumentation: A tutorial. *Argument & Computation 5,* 1, 31–62.

Morgan, M. S. 2017. Narrative ordering and explanation. *Studies in History and Philosophy of Science Part A 62*, 86–97.

Mueller, E. T. 2000. Prospects for in-depth story understanding by computer. *Cognitive Systems Research 23*, 307–340.

Mueller, E. T. 2003. Story understanding through multi-representation model construction. In *Proc. of the HLT-NAACL 2003 Workshop on Text Meaning*, vol. 9. HLT-NAACL-TEXTMEANING. Edmonton, Canada, 31 May 2003, Association for Computational Linguistics, Stroudsburg, PA, USA, 46–53.

Mueller, E. T. 2006. Story understanding. In *Encyclopedia of Cognitive Science*, Lynn Nadel, Ed. John Wiley & Sons, Ltd.

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

Mueller, E. T. 2015. *Commonsense Reasoning: An Event Calculus Based Approach*, 2nd ed. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Ng, H. T., Teo, L. H. and Kwan, J. L. P. 2000. A machine learning approach to answering questions for reading comprehension tests. In *Proc. of the 2000 Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora: Held in Conjunction with the 38th Annual Meeting of the Association for Computational Linguistics*, vol. 13. EMNLP '00. Hong Kong, 7–8 Oct. 2000, Association for Computational Linguistics, Stroudsburg, PA, USA, 124–132.

Pansanato, L., Rivolli, A. and Pereira, D. 2015. An evaluation with web developers of capturing user interaction with rich Internet applications for usability evaluation. *International Journal of Computer Science and Application 4*, 2, 51–60.

Prince, G. 2003. *A Dictionary of Narratology*. University of Nebraska Press.

Rahwan, I. and Simari, G. R. 2009. *Argumentation in Artificial Intelligence*, 1st ed. Springer Publishing Company, Incorporated.

Rick Altman. 2008. *A Theory of Narrative*. Columbia University Press.

Ricoeur, P. 1980. Narrative time. *Critical Inquiry 7,* 1, 169–190.

Riloff, E. and Thelen, M. 2000. A rule-based question answering system for reading comprehension tests. In *Proc. of the 2000 ANLP/NAACL Workshop on Reading Comprehension Tests As Evaluation for Computer-based Language Understanding Systems*, vol. 6.

https://doi.org/10.1017/S1471068418000443 Published online by Cambridge University Press

ANLP/NAACL-ReadingComp '00. Seattle, Washington, USA, 4 May 2000, Association for Computational Linguistics Stroudsburg, PA, USA, 13–19.

Rodosthenous, C. and Michael, L. 2016. A hybrid approach to commonsense knowledge acquisition. In *Proc. of the 8th European Starting AI Researcher Symposium*, The Hague, Holland, 29–30 Aug. 2016, D. Pearce and H. S. Pinto, Eds. IOS Press, 111–122.

Roth, P. A. 1989. How narratives explain. *Social Research 56,* 2, 449–478.

Ryan, M.-L. 2007. Toward a definition of narrative. In *The Cambridge Companion to Narrative*, D. Herman, Ed. Cambridge University Press, 22–35.

Safranski, K. 2017. Codiad Web Based, Cloud IDE. URL: http://codiad.com/. [Accessed on April 18, 2017].

Speer, R. and Havasi, C. 2013. *ConceptNet 5: A Large Semantic Network for Relational Knowledge*. Springer, Berlin, Heidelberg, 161–176.

Toni, F. 2014. A tutorial on assumption-based argumentation. *Argument & Computation 5,* 1, 89–117.

UzZaman, N., Llorens, H., Derczynski, L., Verhagen, M., Allen, J. and Pustejovsky, J. 2013. SemEval-2013 Task 1: TempEval-3: Evaluating time expressions, events, and temporal relations. In *Second joint Conference on Lexical and Computational Semantics (* SEM) 2,* SemEval, 1–9.

Velleman, J. D. 2003. Narrative explanation. *The Philosophical Review 112,* 1, 1–25.

von Ahn, L. and Dabbish, L. 2008. Designing games with a purpose. *Communications of the ACM 51,* 8, 57.

Wellner, B., Ferro, L., Grieff, W. and Hirschman, L. 2006. Reading comprehension tests for computer-based understanding evaluation. *Natural Language Engineering 12,* 4 (Dec.), 305–334.

Wharton, C., Rieman, J., Lewis, C. and Polson, P. 1994. *The Cognitive Walkthrough Method: A Practitioner's Guide*. Tech. rep., New York, NY, USA.

Wielemaker, J., Lager, T. and Riguzzi, F. 2015. SWISH: SWI-Prolog for sharing. In *1st International Workshop on User-Oriented Logic Programming (IULP2015)*.

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

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

Zhang, X., Zhao, J. and LeCun, Y. 2015. Character-level convolutional networks for text classification. In *Advances in Neural Information Processing Systems 28*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, Eds. Curran Associates Inc., 649–657.

https://doi.org/10.1017/S1471068418000443 Published online by Cambridge University Press