Electronic Communications of the EASST
Volume 079 (2020)

Interactive Workshop on the Industrial Application of Verification and Testing ETAPS 2020 Workshop (InterAVT 2020)

Parsing BDD Stories for Automated Verification of Software Artefacts

Thiago Rocha Silva and Brian Fitzgerald

8 Pages

Guest Editors: Stylianos Basagiannis, Goetz Botterweck, Anila Mjeda
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Parsing BDD Stories for Automated Verification of Software Artefacts

Thiago Rocha Silva, Brian Fitzgerald

Lero - The Irish Software Research Centre
University of Limerick (UL)
Limerick, Ireland.
{thiago.silva,brian.fitzgerald}@lero.ie

Abstract: This position paper reports on our ongoing developments towards the automated verification of software artefacts by parsing sentences on Behaviour-Driven Development (BDD) stories. The solution we propose is based on different strategies for analysing the consistency of user requirements specified in BDD stories on task models, graphical user interfaces (GUIs), GUI prototypes, and domain models. We illustrate our solution through concrete examples and discuss its challenges and limitations.

Keywords: Behaviour-Driven Development (BDD), User Requirements Assessment, Software Artefacts, Automated Verification.

1 Introduction

In recent years, User Stories [1] have been widely adopted, especially by agile methods, as an artefact allowing to specify user requirements in a simple, understandable, but yet automatable semi-structured natural language favouring effective communication between the different stakeholders. The use of scenarios in User Stories additionally provides, in a single artefact, the requirements specification along with the set of acceptance criteria which is required to assess whether the system behaves in accordance with such requirements. This strategy has been explored by agile methodologies such as Behaviour-Driven Development (BDD) which gave rise to a particular instance of User Stories, the so-called BDD stories [2].

```
Title (one line describing the story)
Narrative:
As a [role], I want [feature], So that [benefit]
Scenario n: [title]
Given [context], When [event], Then [outcome]
```

Fig. 1. "BDD story" template [3].

A BDD story (Figure 1) is described with a title, a narrative and a set of scenarios representing the acceptance criteria. The title provides a general description of the story, referring to a feature this story represents. The narrative describes the referred feature in terms of the role that will benefit from the feature, the feature itself, and the benefit it will bring to the business. The acceptance criteria are defined through a set of scenarios, each one with a title and three main clauses: *"Given"* to provide preconditions for the scenario, *"When"* to describe an event that will trigger the scenario and *"Then"* to present outcomes that might be checked to verify the proper behaviour of the system. Each one of these clauses can include an *"And"* statement

to provide multiple contexts, events and/or outcomes. Each statement in this representation is called "step".

By specifying the system's expected behaviour through the use of examples, BDD stories carry important information about the system's domain and about how the user is expected to interact with this system. This position paper reports on our developments towards a tool-supported approach to parse these BDD stories and gather information to be automatically verified and assessed on different software artefacts in order to ensure consistency between them. Our motivation for this work resides in the fact that manual verification and software inspections are still the first approaches to verify the consistency between user requirements and such artefacts [4], although manually ensuring the consistency of software specification and its artefacts every time a requirement is introduced and/or modified is extremely time-consuming and highly error-prone. Promoting automated verification is, therefore, a key factor to support assessment in an ever-changing environment.

To achieve this goal, we rely on a previously developed ontology for describing common interactive behaviours on BDD stories [5], [6] which is currently used to support the assessment of user interface design artefacts (task models, GUI prototypes and web GUIs). We have been working on an extension of this approach to also cover the assessment of domain models, such as class diagrams. The final solution is expected to be integrated as an Eclipse plugin which would allow both the specification of the BDD stories and the upload of the respective artefacts being considered for assessment. That would allow a more straightforward integration with other Eclipse-based modelling tools.

The next sections of this paper briefly illustrate, through concrete examples, the solution we propose for assessment on the different covered artefacts, besides discussing its challenges and limitations.

2 Automated Verification of Software Artefacts

The strategy we propose for automated verification of software artefacts [7] has been evolving since 2016 [8]–[11] and is based on an analysis, with the support of an ontology, of different elements specified in the BDD stories' sentences depending on the artefacts being considered for assessment. Figure 2 illustrates the direct relationship between different elements from the BDD stories and the specific elements we verify in the targeted artefacts. In the example, a scenario for successfully withdrawing money from a current account is presented as a user requirement.

The scenario states, in the user's point of view and by means of an example, the resultant effect of withdrawing a certain amount of money. In the figure, the domain-specific behaviour "withdraw" is being detailed in the 5 interaction steps required to effectively conclude the respective action on the user interface of a given system. Thus, to complete the action of withdrawing money in a web system, for example, the user needs to reach the interface in which the transaction types are enlisted, select that he wants to withdraw money, select the current account as the type of account, inform the amount he wants to withdraw, and finally validate the operation by clicking on a button.

Taking the example presented in Figure 2, from the domain-specific behaviours *"Given my balance is $1000"* and *"Then my new balance will be $900"*, *balance* can be identified as an attribute of the class *Account*, and from the behaviour *"When I withdraw $100"*, *withdraw* can be identified as an operation of the class *Account*. The set of interaction steps provides more information to be assessed on the other artefacts. For the first one (*"When I go to 'Transaction*

Type'"), *Transaction* can be identified as a class, and *Type* as an attribute. For this step, a corresponding task can also be identified in the task model (*"Go to Transaction Type"*), as well as the interaction element *Browser Window* in the GUI. For the second step (*"And I select 'Withdraw Money'"*), *Withdraw* can be identified as an operation of the class *Account*, and *Money* as the argument passed when calling it. A corresponding task can also be identified in the task model (*"Select Withdraw Money"*), as well as the interaction element *Vertical Tabs* in the GUI. The other steps can be analysed in a similar manner. The identification of useful elements for assessment on each targeted artefact is discussed hereafter.

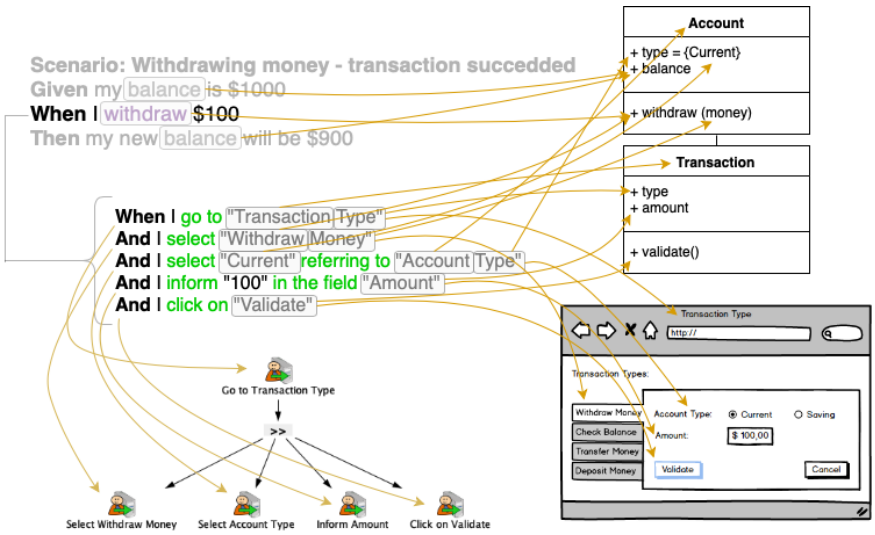

Fig. 2. Tracing the elements from the steps to check the consistency with different artefacts.

Task Models. Task Models (TM) provide a goal-oriented description of interactive systems. Tasks can be specified at various abstraction levels, describing an activity that has to be carried out to fulfil the user's goals [12]. By manipulating task models, we can obtain scenarios that represent the valid interaction paths in the system. This characteristic is particularly useful when identifying test scenarios for the system.

Step of Scenario	Task Name
~~When I~~ set ~~"Valid Departure Date" in the field~~ "Departure Date~~"~~	Set Departure Date

Fig. 3. Formatting rule for assessing interaction steps and tasks.

For assessing task models, we propose a sentence analysis of each interaction step (covered by the ontology) in the BDD stories following a pattern to identify potential tasks to be verified in the model. As task models are designed to support the multiple paths that users may accomplish to perform their tasks, assessing such models involves initially extracting the possible scenarios that are supposed to be tested. The equivalence of steps in BDD stories and tasks in scenarios extracted from task models is supported by the aforementioned ontology which encompasses a formatting rule as exemplified in Figure 3. Our tool applies such a rule

in order to verify whether a behaviour described in a step has an equivalent task to model it in the task model.

This rule aims to eliminate unnecessary components of the step that do not need to be present in the task name. In the example, the component *"When"* refers to the transition in the state machine which is not addressed in a task model. The subject *"I"* signalizes that is the user who performs the task. Tasks models encompass the definition of user role, so the statement *"I"* refers to any users that might correspond to the role assigned to the task model. The verb *"set"* indicates the action that will be performed by the user, so it begins naming the task in the task model. The value *"Valid Departure Date"* indicates the test data domain that will be used to perform and test the task (in this example, any valid value for a departure date of a flight). This is an information which is not present in the task name. The sentence complement *"in the field"* just signalizes that an interaction element (a *"field"*) will be referenced in the sequence. Finally, the target field *"Departure Date"* indicates the name of the interaction element that will be affected by this task, so it composes the final name of the task to be assessed on the task model.

After getting the name of the task to be assessed, we search for that task in the set of scenarios extracted from the task models and if it is found, we evaluate the position in which it has been found in the TM scenario comparing to the position of the corresponding step in the BDD scenario. Thus, we consider there is an inconsistency when (*i*) we do not find a corresponding task in the TM scenario to match the name of the corresponding step in the BDD scenario; or (*ii*) we find a corresponding task in the TM scenario but at a different position of the corresponding step in the BDD scenario.

Graphical User Interfaces (GUIs) and GUI Prototypes. A GUI prototype is an early representation of a user interface. Prototypes are often used in an iterative design process where they are refined and become more and more close to the final GUI through the identification of user needs and constraints. By running simulations on prototypes, we can determine and evaluate potential scenarios that users can perform in the system [13]. Full-fledged GUI versions are the source for acceptance testing and are used by users and other stakeholders to assert whether or not features can be considered as done.

For assessing GUIs (both prototype and final versions), we propose a sentence analysis of each interactive step (covered by the ontology) in the BDD stories identifying (*i*) the interactive behaviour specified, and (*ii*) the affected interaction element. Each step is analysed as follows:

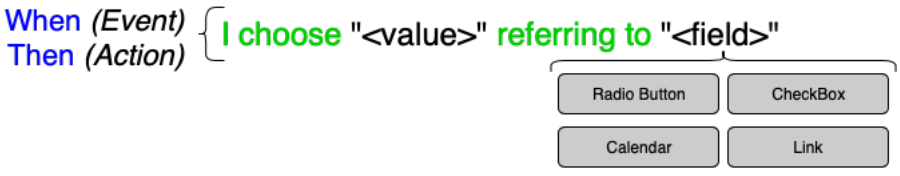

Fig. 4. Structure of an interactive behaviour as specified in the ontology.

An extensive set of interactive behaviours is modelled in the ontology describing how users are supposed to interact with the systems whilst manipulating graphical elements of the GUI. An example of behaviour specification is illustrated by Figure 4. The specification of behaviours encompasses when the interaction can be performed (using *"Given"*, *"When"*

and/or *"Then"* clauses – which corresponds to the context-event-action elements of a state machine), and which graphical interaction elements can be affected. Altogether, behaviours and interaction elements are used to implement the test of an expected system behaviour. In the example presented in the figure, the step *"I choose '<value>' referring to '<field>'"* manipulates a *"<value>"*, which is associated with some test data; and a *"<field>"*, which refers to the interaction element supported by this behaviour, i.e., in this case, a Radio Button, a CheckBox, a Calendar, or a Link. Thus, we consider there is an inconsistency when steps are specified using interactive behaviours that are semantically inconsistent with the affected interaction element on the GUI, such as a selection to be made on a button, for example.

Domain Models. A domain model is a visual representation of conceptual classes or real-world objects in a domain of interest [14]. Class diagrams are among the widest used type of domain models in software engineering. It describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects [15]. Our strategy for assessing class diagrams as domain models is based on parsing the sentences of a BDD story trying to semantically identify elements that should be present in a class diagram. For that, we follow existing heuristics proposed by different works in the literature on natural language processing for requirements engineering and software modelling.

Table 1. Heuristics for parsing natural language requirements (adapted from Lucassen et al. [16])

Rule head (if)	Rule tail (then)
Entities	
Noun	Potential entity
Common noun	Entity
Sentence subject	Entity
Compound noun	Take compound together to form entity
Gerund	Entity
Non-hierarchical relationships	
Verb	Potential relationship
Transitive verb	Relationship
Verb (phrase) linking the subject and an object	Relationship
Verb followed by preposition	Relationship including preposition
Noun–noun compound	Non-hierarchical relationship between prefix and compound
Hierarchical relationships	
Verb "to be"	Subjects are children of parent object
Head of noun–noun compound	IS-A relationship between compound and head

Rule head (if)	Rule tail (then)
Attributes	
Adjective	Attribute of noun phrase main
Adverb modifying a verb	Relationship attribute
Possessive apostrophe	Entity attribute
Genitive case	Entity attribute
Verb ''to have''	Entity attribute
Specific indicators (e.g., "*number*", "*date*", "*type*", ...)	Entity attribute
Object of numeric/algebraic operation	Entity attribute
Cardinality	
Singular noun (+definite article)	Exactly 1
Indefinite article	Exactly 1
Part in the form "More than X"	X..*
Indicators *many*, *each*, *all*, *every*, *some*, *any*	??..*

Most of these works rely on Stanford Parser [17] and WordNet [18]. The Stanford Parser parses sentences in different languages and returns a phrase structure tree (PST) representing the semantic structure of the sentence. WordNet is a large lexical database of English which

groups nouns, verbs, adjectives, and adverbs into sets of cognitive synonyms, each representing a lexicalized concept. By using these tools, Soeken et al. [19] propose a specific set of heuristics for BDD stories in order to identify class diagram elements by parsing their sentences. The authors claim that (*i*) *regular nouns* in sentences usually are realized as objects in the system, and therefore, they can be represented by classes; (*ii*) *adjectives* usually provide further information about the respective objects. Thus, they can be represented by attributes of classes; and (*iii*) *verbs* usually describe actions in a scenario and can therefore be represented by operations of classes. Additionally, they provide information about when an operation is called and by whom.

Other authors have provided more refined heuristics to identify conceptual elements from general requirements in natural language. A comprehensive list of these heuristics is presented in Table 1 adapted from Lucassen et al. [16]. We are currently analysing such heuristics to identify to which extent they fully apply for parsing BDD stories as well. We are also investigating the tools Visual Narrator [16] and ReDoMEx [20] which apply a given subset of these heuristics to generate models. Visual Narrator generates an ontology as a conceptual model from ordinary User Stories, and ReDoMEx generates a class diagram as a domain model from, according to the authors, *unrestricted* natural language requirements. After identifying the conceptual elements, the idea is to point out an inconsistency whenever such elements cannot be found in the class diagrams under assessment.

3 Challenges and Conclusions

We have conducted preliminary case studies with this approach using task models [21], GUI prototypes [22] and final GUIs [23]. Below, we summarize the main challenges, limitations and lessons we have learned so far.

Comprehension of BDD stories. As this approach is expected to benefit a wide range of stakeholders involved with software requirements, verification and testing, one of the main challenges is to ensure that BDD stories are well understood and effectively used by the different stakeholders involved in the project. In a previous study [24], we have preliminary evaluated (*i*) the comprehension of the BDD story template by potential Product Owners (POs), and (*ii*) their ability to effectively use the set of predefined interactive behaviours to specify their own user requirements. Although the participants have followed different specification strategies, we observed an overall high level of adherence to the proposed set of interactive behaviours. The results also pointed out a wide use of domain-dependent behaviours, with the interactive behaviours defined by the ontology being, to some extent, reproduced by the participants even without prior training in the adopted vocabulary. Further studies, however, are still needed to evaluate how useful the proposed vocabulary is to specify user requirements for real software projects in industry involving different stakeholders.

Relevance of artefact maintenance. This approach benefits from the independence for assessing artefacts, i.e. the assessment can be conducted in an independent manner, for example, only in a subset of the artefacts under development or being considered at a given time. In theory, the approach is also well suited to run within any macro software development process, but the maintenance and evolution of the targeted artefacts must make sense in the context of the project. We acknowledge that for some projects, especially in the agile context, artefacts resultant from modelling activities are only used to clarify and agree upon a common understanding of user requirements and are not kept or evolved in practice. Naturally, for these cases, this approach has limited value.

Flexibility and artefact coverage. So far, we are only covering the assessment of task models modelled with the HAMSTERS[1] notation and tool, GUI prototypes designed with the Balsamiq[2] tool, and web GUIs developed under whatever technology for the presentation layer. The approach, however, is domain-independent, i.e. the interactive behaviours described in the ontology (such as clicks, selections, etc.) are the same regardless of the software business domain, and the architecture is flexible enough to accommodate new notations and tools in the future. A limitation is that even with the ontology mapping synonyms for some specific behaviours, it does not provide any kind of semantic interpretation, i.e. the behaviours must be specified exactly as they were defined. At first glance, nonetheless, the restricted vocabulary seems to bring less flexibility to designers, testers, and requirements engineers, but at the same time, it establishes a common vocabulary, avoiding the typical problems of miscommunication, ambiguity, and incompleteness in requirements and testing specifications.

Variety of inconsistencies identified. We identified a wide range of inconsistency problems when running the approach in preliminary case studies. While simple inconsistencies such as differences in names of tasks and fields are easy to be solved, conflicts between specification and modelling along with different specification strategies for task models compose a more critical group of problems and must be prioritized. Concerning the GUIs, the presence of semantically inconsistent elements, the presence of more than one element to represent the same field, and fields already filled-in on web GUIs are also critical groups of problems and denotates inconsistencies that exposes important design errors. Finally, we are still working on case studies to evaluate our strategy for assessing domain models.

Acknowledgments

This work was supported with the financial support of the Science Foundation Ireland grant 13/RC/2094 and has also received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 754489.

References

[1] M. Cohn, *User Stories Applied for Agile Software Development*. Addison-Wesley, 2004.

[2] D. North, "Introducing BDD," *Better Software*, 2006.

[3] D. North, "What's in a Story?," 2019. [Online]. Available: https://dannorth.net/whats-in-a-story/.

[4] M. Chechik and J. Gannon, "Automatic Analysis of Consistency between Requirements and Designs," *IEEE Trans. Softw. Eng.*, vol. 27, no. 7, pp. 651–672, 2001, doi: 10.1109/32.935856.

[5] T. R. Silva, J.-L. Hak, and M. Winckler, "A Behavior-Based Ontology for Supporting Automated Assessment of Interactive Systems," in *Proceedings - IEEE 11th International Conference on Semantic Computing, ICSC 2017*, 2017, pp. 250–257, doi: 10.1109/ICSC.2017.73.

[6] T. R. Silva, J.-L. Hak, and M. Winckler, "A Formal Ontology for Describing Interactive Behaviors and Supporting Automated Testing on User Interfaces," *Int. J. Semant. Comput.*, vol. 11, no. 04, pp. 513–539, 2017, doi: 10.1142/S1793351X17400219.

[7] T. R. Silva, M. Winckler, and H. Trætteberg, "Extending Behavior-Driven Development for Assessing User Interface Design Artifacts," in *Proceedings of the International Conference on Software Engineering and Knowledge Engineering, SEKE*, 2019, pp. 485–488, doi: 10.18293/SEKE2019-054.

[8] T. R. Silva, J.-L. Hak, and M. Winckler, "An Approach for Multi-Artifact Testing Through an

[1] https://www.irit.fr/recherches/ICS/softwares/hamsters/

[2] https://balsamiq.com

Ontological Perspective for Behavior-Driven Development," *Complex Syst. Informatics Model. Q.*, no. 7, pp. 81–107, 2016, doi: 10.7250/csimq.2016-7.05.

[9] T. R. Silva, J.-L. Hak, and M. Winckler, "Testing Prototypes and Final User Interfaces Through an Ontological Perspective for Behavior-Driven Development," in *6th International Working Conference on Human-Centred Software Engineering, and 8th International Working Conference on Human Error, Safety, and System Development (HCSE 2016 and HESSD 2016)*, vol. 9856, 2016, pp. 86–107.

[10] T. R. Silva, "Definition of a Behavior-Driven Model for Requirements Specification and Testing of Interactive Systems," in *Proceedings - 2016 IEEE 24th International Requirements Engineering Conference, RE 2016*, 2016, pp. 444–449, doi: 10.1109/RE.2016.12.

[11] T. R. Silva and M. Winckler, "A Scenario-Based Approach for Checking Consistency in User Interface Design Artifacts," in *IHC'17, Proceedings of the 16th Brazilian Symposium on Human Factors in Computing Systems*, 2017, vol. 1, pp. 21–30, doi: 10.1145/3160504.3160506.

[12] F. Paternò, C. Santoro, L. D. Spano, and D. Raggett, "W3C, MBUI - Task Models," 2017. [Online]. Available: http://www.w3.org/TR/task-models/.

[13] M. Beaudouin-Lafon and W. E. Mackay, "Prototyping Tools and Techniques," in *Prototype Development and Tools*, 2000, pp. 1–41.

[14] C. Larman, *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development*. Addison Wesley Professional, 2004.

[15] I. Jacobson, G. Booch, and J. Rumbaugh, *The Unified Software Development Process*, 1st ed. Addison-Wesley Professional, 1999.

[16] G. Lucassen, M. Robeer, F. Dalpiaz, J. M. E. M. van der Werf, and S. Brinkkemper, "Extracting conceptual models from user stories with Visual Narrator," *Requir. Eng.*, vol. 22, no. 3, pp. 339–358, 2017, doi: 10.1007/s00766-017-0270-1.

[17] M.-C. de Marneffe, B. MacCartney, and C. D. Manning, "Generating Typed Dependency Parses from Phrase Structure Parses," in *Proceedings of the Fifth International Conference on Language Resources and Evaluation, LREC 2006*, 2006, pp. 449–454.

[18] G. A. Miller, "WordNet: A Lexical Database for English George A. Miller," *Commun. Acm*, vol. 38, no. 11, pp. 39–41, 1995.

[19] M. Soeken, R. Wille, and R. Drechsler, "Assisted Behavior Driven Development Using Natural Language Processing," in *TOOLS Europe 2012*, 2012, vol. 7304 LNCS, pp. 269–287.

[20] C. Arora, M. Sabetzadeh, L. Briand, and F. Zimmer, "Extracting Domain Models from Natural-Language Requirements: Approach and Industrial Evaluation," in *Proceedings - 19th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, MODELS 2016*, 2016, pp. 250–260, doi: 10.1145/2976767.2976769.

[21] T. R. Silva, M. Winckler, and H. Trætteberg, "Ensuring the Consistency between User Requirements and Task Models: A Behavior-Based Automated Approach," *Proc. ACM Human-Comput. Interact.*, vol. 4, no. EICS, pp. 77:1–32, 2020, doi: 10.1145/3394979.

[22] T. R. Silva, M. Winckler, and H. Trætteberg, "Ensuring the Consistency Between User Requirements and GUI Prototypes: A Behavior-Based Automated Approach," in *Proceedings of the 17th IFIP TC 13 International Conference on Human-Computer Interaction – INTERACT 2019*, 2019, vol. LNCS 11746, pp. 644–665, doi: 10.1007/978-3-030-29381-9_39.

[23] T. R. Silva, M. Winckler, and H. Trætteberg, "Ensuring the Consistency Between User Requirements and Graphical User Interfaces: A Behavior-Based Automated Approach," in *Proceedings of the 19th International Conference on Computational Science and Its Applications – ICCSA 2019*, 2019, vol. LNCS 11619, pp. 616–632, doi: 10.1007/978-3-030-24289-3_46.

[24] T. R. Silva, M. Winckler, and C. Bach, "Evaluating the usage of predefined interactive behaviors for writing user stories: an empirical study with potential product owners," *Cogn. Technol. Work*, May 2019, doi: 10.1007/s10111-019-00566-3.