On the Composability of Behavior Driven Acceptance Tests

Tugkan Tuglular
Department of Computer Engineering
Izmir Institute of Technology
Izmir, Turkey
mail: tugkantuglular@iyte.edu.tr

***Abstract*—This paper proposes a model-based approach for composition of Behavior Driven Acceptance Tests (BDATs) using Event Sequence Graphs (ESGs). ESGs are used to generate test sequences automatically. For the composition process of BDATs, the ESG formalism is extended with tags and the technique called elimination of tags by combination is introduced for tagged ESGs. The proposed approach improves testability of existing behavior driven acceptance test suites. It is validated through a real-life example. The results demonstrate the feasibility of the proposed approach.**

Keywords-model-based testing; event sequence graphs; behavior driven acceptance tests; Gherkin.

I. INTRODUCTION

Behavior Driven Development (BDD) is focused on defining fine-grained specifications of the behavior of the targeted system [1]. In BDD, tests are clearly written using a specific ubiquitous language, such as Gherkin [2]. For developing Behavior Driven Acceptance Tests (BDATs), there are environments like Cucumber [2], which forces testers to use a test template using Gherkin language and environments like Gauge [3], which does not impose any language. The scope of this study is BDATs developed in Gherkin.

Although Gherkin and its scenario template helps test designers in writing test cases, they do not guide test designers in test objectives. The test designer either develops BDATs in an ad-hoc manner or follows rules of thumb such as happy path testing and negative testing. In either case, the test designer is not certain about the completeness or coverage of the BDAT test suite. As a solution, this paper proposes to transform Gherkin scenarios into formal test models, so that the test designer can work on completeness and coverage of BDATs.

The proposed approach assumes that clauses written in Gherkin can be represented by events. In that case, an event-based formal model would fit better to BDATs. Therefore, this paper proposes the use of Event Sequence Graphs (ESGs) for modeling BDATs. To model a BDAT as an ESG, ESGs are extended with tags. This is one of the novelties presented in this paper. Another novelty presented here is the process of finding missing BDATs. To find missing BDATs, the proposed approach follows elimination of tags by combination. After the missing BDATs are completed, an ESG without any tags is obtained. The proposed approach is explained with a running example in Section III. For evaluation, a BDAT test suite is selected from Github^{TM} and the proposed approach is applied to this test suite. The results are shared in Section IV.

The paper is organized as follows: In the next section, the formal definitions of ESGs are given along with examples and figures. The proposed approach is explained in Section III. Section IV gives an evaluation of the proposed approach along with a discussion in Section V. Section VI outlines related work, and the last section concludes the paper.

II. FUNDAMENTALS

A. *Gherkin*

Gherkin uses a set of special keywords to give structure and meaning to executable specifications [2]. It provides the behavior definitions of the intended software not only to product owners and business analysts, but also to developers and testers [4]. Gherkin is a line-oriented language in terms of structure and each line has to be divided by the Gherkin keyword except feature and scenario descriptions [2]. In this paper, some of the Gherkin keywords; namely *Feature*, *Scenario*, *Given*, *When*, *And*, *Then*, are utilized. Throughout the paper, the terms Gherkin scenario, scenario, and BDAT are used interchangeably.

Tests should be independent of each other so that they can be run in any order or even in parallel. This principle is also applied in developing BDATs. So, each BDAT should be run manually or automatically independent of other BDATs. However, they should also be composable so that it will be possible to execute a BDAT after a related one.

B. *Event Sequence Graphs*

A model of the system, which requires the understanding of its abstraction, helps in testing its behavior. A formal specification approach that distinguishes between legal and illegal situations is necessary for acceptance testing. These requirements are satisfied by event sequence graphs [5].

Differing from the notion of finite-state automata, inputs and states are merged in ESG, hence they are turned into "events" to facilitate the understanding and checking the external behavior of the system. Thus, vertices of the ESG represent events as externally observable phenomena, e.g., a user action or a system response. Directed edges connecting two events define allowed sequences among these events [5]. Definitions from 1 to 3 and related examples and explanations along with Figure 1 are taken exactly as they are from [6]-[9].

Definition 1. An *event sequence graph ESG = (V, E, Ξ, Γ)* is a directed graph where $V \neq \emptyset$ is a finite set of vertices (nodes), $E \subseteq V \times V$ is a finite set of arcs (edges), $\Xi, \Gamma \subseteq V$ are finite sets of distinguished vertices with $\xi \in \Xi$, and $\gamma \in \Gamma$, called entry nodes and exit nodes, respectively, wherein $\forall v \in V$ there is at least one sequence of vertices $\langle \xi, v_0, \ldots, v_k \rangle$ from each $\xi \in \Xi$ to $v_k = v$ and one sequence of vertices $\langle v_0, \ldots, v_k, \gamma \rangle$ from $v_0 = v$ to each $\gamma \in \Gamma$ with $(v_i, v_{i+1}) \in E$, for $i = 0, \ldots, k\text{-}1$ and $v \neq \xi, \gamma$.

To mark the entry and exit of an ESG, all $\xi \in \Xi$ are preceded by a pseudo vertex '[' $\notin V$ and all $\gamma \in \Gamma$ are followed by another pseudo vertex ']' $\notin V$. The semantics of an ESG are as follows. Any $v \in V$ represents an event. For two events $v, v' \in V$, the event v' must be enabled after the execution of v iff $(v, v') \in E$. The operations on identifiable components of the GUI are controlled and/or perceived by input/output devices, i.e., elements of windows, buttons, lists, checkboxes, etc. Thus, an event can be a user input or a system response; both of them are elements of V and lead interactively to a succession of user inputs and expected desirable system outputs.

Example 1. For the ESG given in Figure 1: $V=\{a,b,c\}$, $\Xi=\{a\}$, $\Gamma=\{b\}$, and $E = \{(a,b), (a,c),(b,c),(c,b))\}$. Note that arcs from pseudo vertex [and to pseudo vertex] are not included in *E*.

Furthermore, α(initial) and ω(end) are functions to determine the initial vertex and end vertex of an ES, e.g., for ES= $(v_0, \ldots, v_k)$ initial vertex and end vertex are $\alpha(ES)=v_0$, $\omega(ES)=v_k$, respectively. For a vertex $v \in V$, $N^+(v)$ denotes the set of all *successors* of v, and $N^-(v)$ denotes the set of all *predecessors* of v. Note that $N^-(v)$ is empty for an entry $\xi \in \Xi$ and $N^+(v)$ is empty for an exit $\gamma \in \Gamma$.

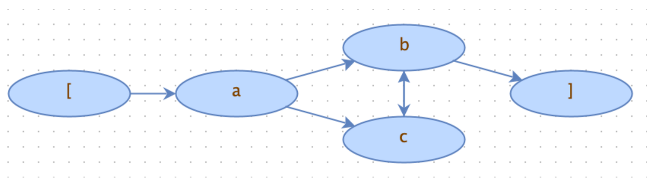

Figure 1. An ESG with a as entry and b as exit and pseudo vertices [,].

Definition 2. Let V, E be defined as in Definition 1. Then, any sequence of vertices $\langle v_0, \ldots, v_k \rangle$ is called an *event sequence* (*ES*) iff $(v_i, v_{i+1}) \in E$, for $i=0, \ldots, k\text{-}1$.

The function *l*(*length*) of an ES determines the number of its vertices. In particular, if $l(ES)=1$ then $ES=(v_i)$ is an ES of length 1. Note that the pseudo vertices [and] are not considered in generating any ESs. Neither are they included in ESs nor considered to determine the initial vertex, end vertex, and length of the ESs. An ES = $\langle v_i, v_k \rangle$ of length 2 is called an *event pair* (EP).

Definition 3. An *ES* is a *complete ES* (or, it is called a *complete event sequence*, *CES*), if $\alpha(ES)=\xi \in \Xi$ is an entry and $\omega(ES)=\gamma \in \Gamma$ is an exit.

A CES may or may not invoke no interim system responses during user-system interaction. If it does not, that means that it consists of consecutive user inputs and only a final system response. CESs represent walks from the entry of the ESG to its exit, realized by the form (initial) user inputs → (interim) system responses → ⋯ (interim) user inputs → (interim) system responses → ⋯ → (final) system response.

III. Proposed Approach

The proposed approach improves completeness of a BDAT test suite and enables coverage-based test sequence generation. With the assumption that Gherkin clauses can be represented by events, the proposed approach suggests the use of ESGs for modeling BDATs. To model a BDAT as an ESG, ESGs are extended with tags. This is explained first in this section. Then, how BDATs are combined using tagged ESGs is presented. After that, elimination of tags by combination process that is used to find missing BDATs is outlined. This section concludes with an example where all BDATs, i.e., original, missing, and additional BDATs, are composed into one ESG without any tags.

A. Representation of BDATs with tagged ESGs

Best practice for Gherkin scenarios is to describe behavior rather than functionality.

A behavior driven acceptance test is a specification of the behavior of the system, which verifies the interactions of the objects rather than their states [10]. A scenario that makes up a BDAT is composed of several steps. A step is an abstraction that represents one of the elements in a scenario which are: contexts, events, and actions [1]. So, a Gherkin scenario template is as follows:

```
Given context
When event
Then action
```

Contexts, events, and actions can be represented by events. A context is formed after a sequence of events. For instance, the line Given I am on the homepage in a scenario indicates that the context is being on the homepage and the user can reach the homepage by a sequence of events. So, we can say that a context is the result of a sequence of events. Sometimes, the sequence of events may be empty. An action is an event or results in an event depending on your standpoint. For instance, the line Then product list is displayed in a scenario is the action of the software, but for the user it is an event.

This paper proposes the use of event sequence graphs for modeling BDATs. To model a BDAT as an ESG, ESGs are extended with tags.

Definition 4. A tagged ESG is an ESG, where a node or vertex may contain a tag instead of an event.

A tagged ESG is useful in transforming Gherkin scenarios or BDATs to ESGs. Contexts and actions are represented by tags and this way, tags become connection or composition points for ESGs. For instance, in the following Scenario cart02, *Given* event is tagged with #productPage and *Then* event is tagged with #shoppingBasket. Its ESG representation is shown in Figure 2.

Scenario: cart02 - Adding a product to cart
Given I am on a product detail page #productPage
When I select the amount
And I click the add to cart button
Then the product is added to my shopping cart #shoppingCart

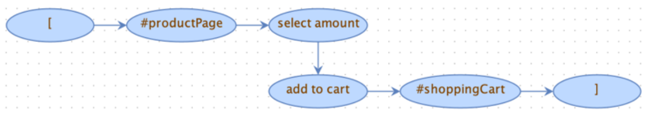

Figure 2. Tagged ESG for Scenario cart02.

Annotating Gherkin clauses with tags and representing BDATs with tagged ESGs enable us to combine BDATs.

B. *Combining two BDATs on tagged ESG*

To combine two BDATs, the following approach is proposed. Ending Gherkin clause can be combined with starting Gherkin clause if they have the same tag. This means two Gherkin scenarios can be run in a sequence. We can connect Scenario cart02 with Scenario check01 presented below, where *Given* event is tagged with #shoppingBasket and *Then* event is tagged with #orderConfirmed. ESG representation of Scenario check01 is shown in Figure 3.

Scenario: check01 - Successful checkout
Given I have added an item to my shopping bag #shoppingCart
When I proceed to the check out
And I enter valid delivery details
And I select a payment method
And I confirm the order
Then I am redirected to the thank you page #orderConfirmed

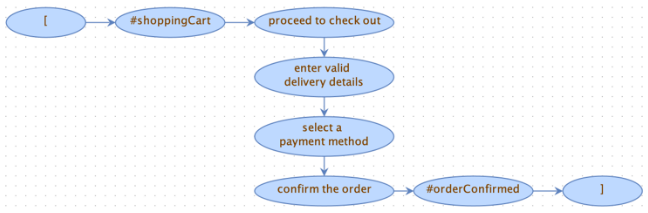

Figure 3. Tagged ESG for Scenario check01.

As seen, tags are used as connection points. Following the approach presented in Section III-A, we can combine these two BDATs on a tagged ESG, since both are represented as a tagged ESG. The resulting tagged ESG is shown in Figure 4.

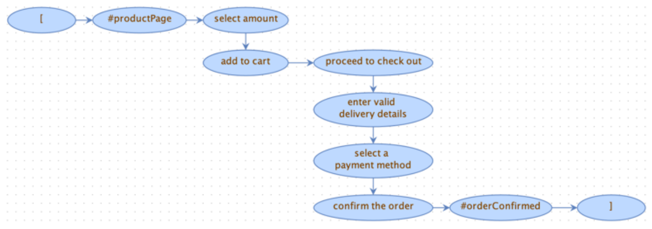

Figure 4. Tagged ESG for combined Scenarios cart02 and check01.

C. *Finding missing BDATs*

To find missing BDATs, elimination by combination is proposed. As seen in Section III-B, once two BDATs are combined using a tag, that tag is eliminated. Therefore, first all possible tagged scenarios or their graphical representations, i.e., tagged ESGs, are combined. It should be noted that a combined tagged ESG may be combined with another simple or combined tagged ESG. The goal is to reach an ESG without any tags, as shown in Figure 5. After all possible combinations are completed, if a tag remained on a tagged ESG indicates that there is a missing BDAT. If there are more than one tag, that may mean more missing BDATs.

For instance, in the following Scenario acc03, *Given* event is tagged with #atHome and *Then* event is tagged with #orderDetail.

Scenario: acc03 - Check orders
Given I am logged in on the site #atHome
When I navigate to my orders
Then I see a list of my orders
And I can open an order to see the order details #orderDetail

This BDAT is the only Gherkin scenario that has the tag #orderDetail. Since there is no match, it indicates that a BDAT that starts with #orderDetail tag is missing. We can complete this missing BDAT as follows:

Scenario: acc10 - Back to order list page
Given #orderDetail
When I press OK button
Then order list page is displayed #orderList

As seen in the running example, elimination by combination shows us clues about completeness of BDATs. The approach proposed here is to check whether all tags are combined. Any tag that is not combined suggests a missing BDAT.

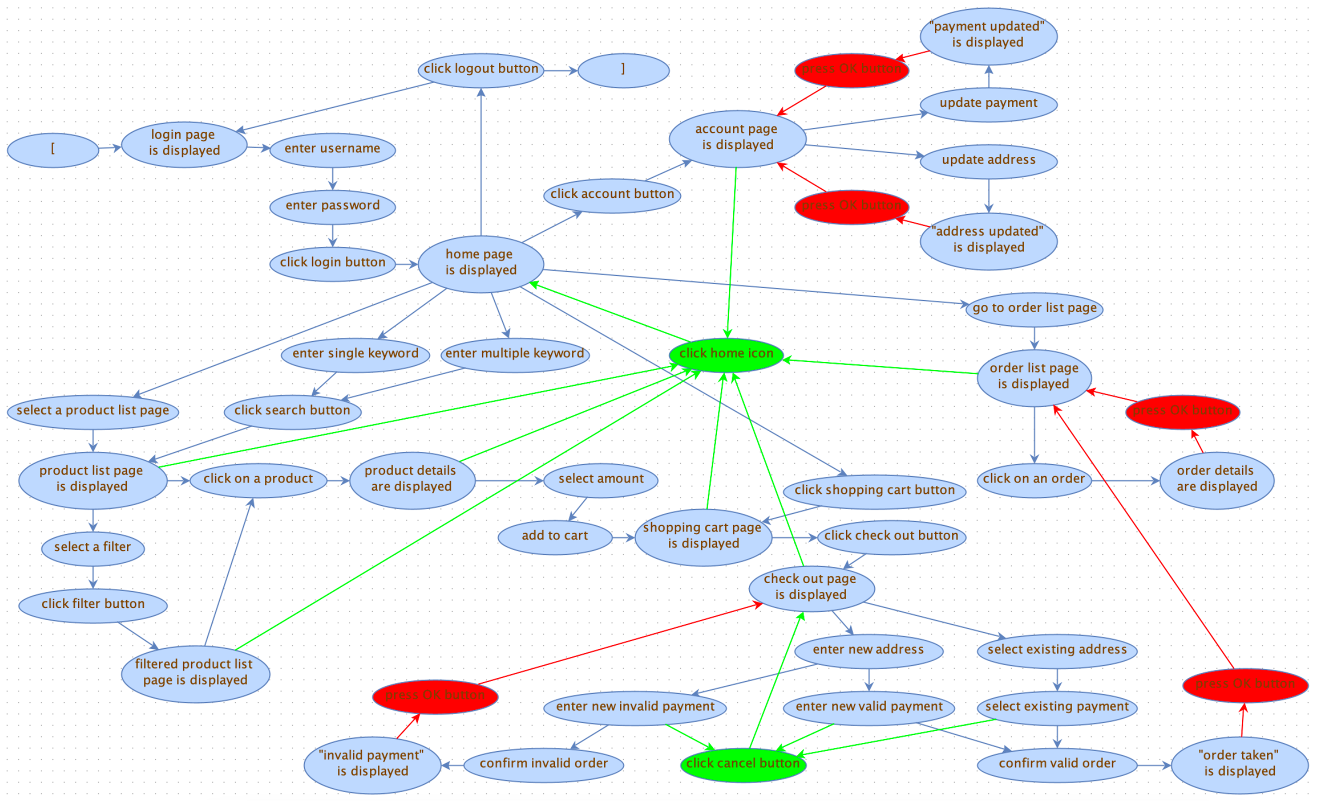

Figure 5. Composed ESG.

D. *Composition of BDATs on tagged ESG*

After completing the missing BDATs and improving existing BDATs, the BDATs are composed on an ESG. The resulting ESG is shown in Figure 5. There are no tags on the resulting ESG, which means that all tags can be eliminated by combination. Elimination by combination enables us to find five missing BDATs, which are drawn in red on the resulting ESG in Figure 5.

Once ESG is ready then CES for edge and for edge-pair coverage can be generated for BDATs. The details of CES generation can be found in [8]. We utilized the TSD tool [11] to generate CES for both coverage criteria. The results are given in the following section.

IV. Evaluation

For evaluation, the proposed approach is applied to an existing test suite for an e-commerce software [12], of which six features out of eight are taken for evaluation. The features locale and newsletter are left. The existing test suite has 15 scenarios, or BDATs, with 64 Gherkin clauses. Clause per scenario ratio is 4.26.

After applying the proposed approach, we end up with 24 BDATs and 85 Gherkin clauses. There are 9 new scenarios but only 5 of them are missing scenarios. The other 4 scenarios are introduced to simplify and standardize some original scenarios. So, clause per scenario ratio is decreased to 3.54 from 4.26. The comparison of before and after the proposed approach is given in Table I. The resulting test suite has the scenarios that are simplified, standardized, and tagged. Moreover, they become composable.

TABLE I. Comparison of Before and After Proposed Approach

Criteria	*Before*	*After*
Number of scenarios	15	24
Number of clauses	64	85
Clause per scenario ratio	4.26	3.54

A further analysis of the resulting ESG shows that event sequences are stuck in the child pages of home page. There is no return to home page from child pages, which means that features of the software cannot be tested in sequence. In addition, it is discovered that there is no scenario about cancellation of the check-out process. Those BDATs, 10 in total, are added in green to the resulting ESG in Figure 5. It should be noted that the graphical representation of BDATs enables us to perform such an analysis. Without tool support, it is very hard for test designers to conduct such analysis on text represented BDATs.

There is another advantage of the proposed approach. Since BDATs are transformed to ESGs and then combined, we have an ESG from which we can automatically generate test sequences, i.e., sequences of BDATs. CES for edge coverage computed by the TSD tool is shown below:

No. of Nodes: 50
No. of Edges: 70
CES with 111 events:
[, login page is displayed, enter username, enter password, click login button, home page is displayed, go to order list page, order list page is displayed, click on an order, order details are displayed, press OK button, order list page is displayed, click home icon, home page is displayed, click shopping cart button, shopping cart page is displayed, click check out button, check out page is displayed, enter new address, enter new invalid payment, confirm invalid order, "invalid payment" is displayed, press OK button, check out page is displayed, enter new address, enter new invalid payment, click cancel button, check out page is displayed, enter new address, enter new valid payment, click cancel button, check out page is displayed, select existing address, select existing payment, click cancel button, check out page is displayed, enter new address, enter new valid payment, confirm valid order, "order taken" is displayed, press OK button, order list page is displayed, click home icon, home page is displayed, enter multiple keyword, click search button, product list page is displayed, select a filter, click filter button, filtered product list page is displayed, click on a product, product details are displayed, select amount, add to cart, shopping cart page is displayed, click home icon, home page is displayed, enter single keyword, click search button, product list page is displayed, click on a product, product details are displayed, click home icon, home page is displayed, select a product list page, product list page is displayed, click home icon, home page is displayed, click account button, account page is displayed, update payment, "payment updated" is displayed, press OK button, account page is displayed, update address, "address updated" is displayed, press OK button, account page is displayed, click home icon, home page is displayed, click shopping cart button, shopping cart page is displayed, click check out button, check out page is displayed, select existing address, select existing payment, confirm valid order, "order taken" is displayed, press OK button, order list page is displayed, click home icon, home page is displayed, select a product list page, product list page is displayed, select a filter, click filter button, filtered product list page is displayed, click home icon, home page is displayed, click shopping cart button, shopping cart page is displayed, click check out button, check out page is displayed, click home icon, home page is displayed, click logout button, login page is displayed, enter username, enter password, click login button, home page is displayed, click logout button,],

CES for edge-pair coverage computed by the TSD tool has a complete event sequence of 224 events. The CES is not given here because of space limitations.

V. DISCUSSION

The proposed approach assumes that Gherkin clauses can be represented by events. This assumption holds for the selected test suite used in the evaluation. Although Gherkin is developed for behavioral description scenarios, it must be shown that all possible Gherkin clauses and scenarios can be represented by events. It may require some transformation. This is left as future work.

The proposed approach shows that through modeling BDATs, it is possible to automatically generate test sequences. UML use case diagrams and activity diagrams can also be used for modeling BDATs and then automatically generate tests. The research in this area is explained in the related work section.

Scalability of the models is an important concern. ESGs allow us to work on some small and modular models through sub-ESGs [6]-[9] like subroutines. The TSD tool is also designed to support sub-ESGs. This way, it is possible to generate manageable large models. Moreover, these sub-ESGs can be flattened into one large ESG if necessary.

One threat to validity is internal validity, which deals with the effects on the evaluation. The selection of BDAT test suite used in evaluation is obtained by searching GitHub repositories. This cannot be considered as random selection. Moreover, the proposed approach is applied to the selected BDAT test suite by the author.

Another threat to validity is external validity, which deals with the generalizability of the results. The evaluation in this study is based on a single BDAT test suite. Although this test suite is developed for e-commerce software, which may represent business software generally, evaluation of other BDAT test suites from different domains with the proposed approach will help generalize the results.

VI. RELATED WORK

Tuglular [13] proposed a model-based approach for feature-oriented testing using Event Sequence Graphs (ESGs). In this approach, ESGs are extended to save state and pass it to the following ESG. This way, tests written for features can be combined on state information. However, capturing state is not always possible for acceptance tests.

UML use case diagrams can also be used for modeling BDATs and then automatically generate tests. Gutierrez et al. [14] proposed an approach for working with Gherkin scenarios using UML use case models. They transform from the UML use case diagrams to the Gherkin plain text syntax. They also developed a tool for running Gherkin scenarios in UML as test cases.

Alferez et al. [15] proposed an approach, named AGAC (Automated Generation of Acceptance Criteria), which supports the automated generation of AC specifications in Gherkin. They used UML use case diagrams and activity diagrams to create specifications, derive acceptance criteria from them, and then generate test cases from derived acceptance criteria.

Kudo et al. [16] proposed the software pattern meta model that bridges requirement patterns to groups of scenarios with similar behaviors in the form of test patterns. This meta model is used to describe the behavior of a requirement pattern through a time executable and easy-to-use language aiming at the automatic generation of test patterns.

Wanderley and da Silveria [17] proposed using a mind model specification, which serves as a basis for transforming the definitions of the scenario and generating a conceptual model represented by a UML class diagram. The mind model functions as a bond that represents the business entities, and enables simple association, aggregation and composition relationships between the entities.

An adjacent area is process discovery in business process management literature. Rozinat and van der Aalst [18] worked on whether event logs conform to the process model and vice versa. They proposed two dimensions of conformance, namely fitness and appropriateness, to be checked along with corresponding metrics. They developed a Conformance Checker within the ProM Framework.

Beschastnikh et al. [19] proposed algorithms for inferring communicating finite state machine models from traces of concurrent systems, and for proving them correct. They also provided an implementation called CSight, which helps developers find bugs.

Pecchia et al. [20] proposed an approach that employs process mining for detecting failures from application logs. Their approach discovers process models from logs; then it uses conformance checking to detect deviations from the discovered models. They were able to quantify the failure detection capability of conformance checking in spite of missing events, and its accuracy with respect to process models obtained from noisy logs [20].

VII. Conclusion

This paper proposes an approach to represent BDATs using ESGs. With the proposed approach, the test designer not only finds and completes missing BDATs, but also combines them to know which BDAT can be executed after which BDAT. When the final composition is supplied to the TSD tool, it automatically generates a test sequence that covers all BDATs. So, the proposed approach improves testability of BDATs.

As future work, we plan to automate the processes explained here and develop a tool. Also as future work, our goal is to enhance the tool with ontologies so semantically related scenarios are easily decoded.

References

[1] M. G. Cavalcante and J. I. Sales, "The Behavior Driven Development Applied to the Software Quality Test," Proc. 14th Iberian Conference on Information Systems and Technologies (CISTI), IEEE, 2019, pp.1–4.

[2] Cucumber Gherkin. https://cucumber.io/docs/gherkin/ reference/. [retrieved: March, 2021].

[3] Gauge. https://gauge.org. [retrieved: March, 2021].

[4] T. Tuglular and S. Şensülün. "SPL-AT Gherkin: A Gherkin Extension for Feature Oriented Testing of Software Product Lines." 2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC). Vol. 2. IEEE, 2019, pp.344–349.

[5] T. Tuglular, F. Belli, and M. Linschulte, "Input contract testing of graphical user interfaces," International Journal of Software Engineering and Knowledge Engineering, 26(02), 2016, pp.183–215.

[6] F. Belli and C. J. Budnik, "Test minimization for human-computer interaction," Applied Intelligence, 26(2), 2007, pp.161–174.

[7] F. Belli, C. J. Budnik, and L. White, "Event based modelling, analysis and testing of user interactions: approach and case study," Software Testing, Verification and Reliability, 16(1), 2006, pp.3–32.

[8] F. Belli and C. J. Budnik, "Minimal spanning set for coverage testing of interactive systems," International Colloquium on Theoretical Aspects of Computing. Springer, Berlin, Heidelberg, 2004, pp.220–234.

[9] T. Tuglular, C. A. Muftuoglu, F. Belli, and M. Linschulte, "Event-based input validation using design-by-contract patterns," Proc. 20th International Symposium on Software Reliability Engineering, ISSRE'09, IEEE Press, 2009, pp. 195–204.

[10] E. Evans, "Domain-Driven Design: Tackling Complexity in the Heart of Software," Addison-Wesley Professional, 2003.

[11] TestSuiteDesigner. http://download.ivknet.de/. [retrieved: March, 2021].

[12] Barzilay, "Example of an ECommerce cucumber web test automation suite". https://github.com/spriteCloud/ecommerce -cucumber-web-test-automation-suite. [retrieved: March, 2021].

[13] T. Tuglular, "Event sequence graph-based feature-oriented testing: A preliminary study," 2018 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C). IEEE, 2018, pp. 580–584.

[14] J. J. Gutiérrez, I. Ramos, M. Mejías, C. Arévalo, J. M. Sánchez-Begines, and D. Lizcano, "Modelling Gherkin Scenarios Using UML," Proc. 26th International Conference on Information Systems Development (ISD), 2017, http://aisel.aisnet.org/isd2014/proceedings2017/ISDMethodol ogies/7.

[15] M. Alferez, F. Pastore, M. Sabetzadeh, L. Briand, and J. R. Riccardi, "Bridging the gap between requirements modeling and behavior-driven development," 2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems (MODELS), IEEE, 2019, pp. 239–249.

[16] T. N. Kudo, R. F. Bulcão-Neto, and A. M. Vincenzi, "A conceptual metamodel to bridging requirement patterns to test patterns," Proc. of the XXXIII Brazilian Symposium on Software Engineering. 2019, pp.155–160.

[17] F. Wanderley and D. S. da Silveria, "A framework to diminish the gap between the business specialist and the software designer," 2012 Eighth International Conference on the Quality of Information and Communications Technology. IEEE, 2012, pp. 199–204.

[18] A. Rozinat and W.M.P. van der Aalst, "Conformance testing: Measuring the fit and appropriateness of event logs and process models," Proc. 4th Business Process Management Workshops, Springer, 2006, pp. 163–176.

[19] I. Beschastnikh, Y. Brun, M.D. Ernst, and A. Krishnamurthy, "Inferring models of concurrent systems from logs of their behavior with CSight," Proc. 36th International Conference on Software Engineering, ACM, 2014, pp. 468–479.

[20] A. Pecchia, I. Weber, M. Cinque, and Y. Ma., "Discovering process models for the analysis of application failures under uncertainty of event logs," Knowledge-Based Systems, 2020, 189: 105054, https://doi.org/10.1016/j.knosys.2019.105054.