Ensuring the Consistency between User Requirements and Task Models: A Behavior-Based Automated Approach

Thiago Rocha Silva, Marco Winckler, Hallvard Trætteberg

Ensuring the Consistency between User Requirements and Task Models: A Behavior-Based Automated Approach

Article *in* Proceedings of the ACM on Human-Computer Interaction · June 2020

DOI: 10.1145/3394979

CITATIONS
3

READS
188

3 authors:

Thiago Rocha Silva
University of Limerick
37 PUBLICATIONS **118** CITATIONS
SEE PROFILE

Marco Winckler
University of Nice Sophia Antipolis
229 PUBLICATIONS **1,539** CITATIONS
SEE PROFILE

Hallvard Trætteberg
Norwegian University of Science and Technology
51 PUBLICATIONS **523** CITATIONS
SEE PROFILE

Ensuring the Consistency between User Requirements and Task Models: A Behavior-Based Automated Approach

THIAGO ROCHA SILVA, Lero - The Irish Software Research Centre, University of Limerick (UL), Ireland
MARCO WINCKLER, SPARKS-I3S, Université Nice Sophia Antipolis (Polytech), France
HALLVARD TRÆTTEBERG, IDI, Norwegian University of Science and Technology (NTNU), Norway

Evaluating and ensuring the consistency between user requirements and modeling artifacts is a long-time issue for model-based software design. Conflicts in requirements specifications can lead to many design errors and have a decisive impact on the quality of systems under development. This article presents an approach based on Behavior-Driven Development (BDD) to provide automated assessment for task models, which are intended to model the flow of user and system tasks in an interactive system. The approach has been evaluated by exploiting user requirements described by a group of experts in the domain of business trips. Such requirements gave rise to a set of BDD stories that have been used to automatically assess scenarios extracted from task models that were reengineered from an existing web system for booking business trips. The results have shown our approach, by performing a static analysis of the source files, was able to identify different types of inconsistencies between the user requirements and the set of task models analyzed.

CCS Concepts: • **Human-centered computing** → *User centered design*; • **Software and its engineering** → *Software verification and validation.*

Additional Key Words and Phrases: Behavior-Driven Development (BDD); User Stories; Task Models; Automated Requirements Assessment

ACM Reference Format:
Thiago Rocha Silva, Marco Winckler, and Hallvard Trætteberg. 2020. Ensuring the Consistency between User Requirements and Task Models: A Behavior-Based Automated Approach. *Proc. ACM Hum.-Comput. Interact.* 4, EICS, Article 77 (June 2020), 32 pages. https://doi.org/10.1145/3394979

1 INTRODUCTION

Modeling is recognized as a crucial activity to manage the abstraction and the inherent complexity of developing software systems. Several aspects of information, from the macro business goals until the most detailed information about user tasks, are taken into account while modeling. The outcomes of modeling activities are registered by means of software artifacts. Artifacts encode a particular interpretation of a problem situation and a particular set of solutions for the perceived problem [10].

Requirements and artifacts are also expected to evolve along the project according to the users' changing perception about their own needs. In iterative processes, the cycle of producing and evolving requirements artifacts permeates all the phases of system development, from requirements

Authors' addresses: Thiago Rocha Silva, thiago.silva@lero.ie, Lero - The Irish Software Research Centre, University of Limerick (UL), Limerick, Ireland; Marco Winckler, winckler@unice.fr, SPARKS-I3S, Université Nice Sophia Antipolis (Polytech), Sophia Antipolis, France; Hallvard Trætteberg, hal@ntnu.no, IDI, Norwegian University of Science and Technology (NTNU), Trondheim, Norway.

2573-0142/2020/6-ART77 $15.00
https://doi.org/10.1145/3394979

and business analysis until software testing. Artifacts are useful even after the software development has finished, for software maintenance and evolution purposes, since they encompass models describing the record of implemented requirements, which strategies were used to implement the features, how the system architecture has been structured, and so on. Therefore, requirements should be described in a consistent way across multiples versions of artifacts which is a quite challenging activity [51]. Requirements specifications should not, for example, describe a given requirement in a User Story that is conflicting with its representation in a task model.

Behavior-Driven Development (BDD) [5] has stood out in the software engineering community as an effective approach to provide automated acceptance testing by specifying natural language user requirements and their tests in a single textual artifact. BDD benefits from a requirements specification based on User Stories (US) [7] which are easily understandable for both technical and non-technical stakeholders. In addition, User Stories allow specifying "executable requirements", i.e. requirements that can be directly tested from their textual specification. Despite its benefits providing automated testing of user requirements, BDD and other testing approaches focus essentially on assessing fully implemented versions of the system. Since long time ago, it is a peaceable argument that providing early assessment is very helpful for detecting errors before making strong commitments with the software implementation [49]. Nonetheless, as far as early artifacts such as task models are concerned, current approaches offer no support for automated assessment.

Motivated by such a gap, this article presents an approach based on BDD and User Stories to support the automated assessment of the consistency between user requirements and task models. The common-ground of concepts for describing the artifacts and verifying their consistency is provided by means of an ontology [35], [38]. The following sections present the foundations and related work, the proposed approach with its technical implementation, and the results we got by assessing the reengineered task models from an existing web system to book business trips.

2 FOUNDATIONS

2.1 Behavior-Driven Development and User Stories

According to Smart [41], BDD is a set of software engineering practices designed to help teams focus their efforts on identifying, understanding, and building valuable features that matter to businesses. BDD practitioners use conversations around concrete examples of system behavior to help understand how features will provide value to the business. BDD encourages business analysts, software developers, and testers to collaborate more closely by enabling them to express requirements in a more testable way, in a form that both the development team and business stakeholders can easily understand. BDD tools can help turn these requirements into automated tests that help guide the developer, verify the feature, and document the application.

BDD specification is based on User Stories and scenarios which allows to specify executable requirements and test specifications by means of a Domain-Specific Language (DSL) provided by Gherkin [15]. User Stories were firstly proposed by Cohn [7]. BDD added the description of scenarios to the User Stories in order to provide, in a single artifact, the requirements specification along with the set of acceptance criteria which is required to assess whether the system behaves in accordance with such requirements. North [27] proposed a template for this particular kind of User Story and named it as "BDD story" (Figure 1).

According to this template, a BDD story is described with a title, a narrative and a set of scenarios representing acceptance criteria, giving concrete examples about what need to be tested to consider a given feature as done. 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

```
Title (one line describing the story)

Narrative:
As a [role]
I want [feature]
So that [benefit]

Acceptance Criteria: (presented as Scenarios)

Scenario 1: Title
Given [context]
  And [some more context]...
 When [event]
 Then [outcome]
  And [another outcome]...

Scenario 2: ...
```

Fig. 1. "BDD story" template [27]

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 behavior 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 a "*step*".

2.2 Task Models

Task models (TM) provide a goal-oriented description of interactive systems but avoiding the need for the level of detail required for a full description of the user interface. Tasks can be specified at various abstraction levels, describing an activity that has to be carried out to fulfil the user's goals. By modeling tasks, designers are able to describe activities in a fine granularity, for example, covering the temporal sequence of tasks to be carried out by the user or system, as well as any preconditions for each task [30]. The use of task models serves as multiple purposes such as better understanding the application under development (and in particular its use), being a "record" of multidisciplinary discussions between multiple stakeholders, helping the design, the usability evaluation, the performance evaluation, and the user in performing the tasks (acting as a contextual help). Task models are also useful as documentation of requirements both related with content and structure.

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. Being scenarios a description of a specific use in a specific context, and task models descriptions of possible activities and their relationships, scenarios support task development while task models can support the identification of scenarios.

2.2.1 An Overview of HAMSTERS. Human-centered Assessment and Modeling to Support Task Engineering for Resilient Systems (HAMSTERS) [25] is a notation inspired by other existing ones for task modeling, especially CTT [29]. HAMSTERS includes extensions such as preconditions associated with task executions, data flow across task models, and more detailed interactive tasks. HAMSTERS' models can be edited and simulated in a dedicated environment which also provides a dedicated API for observing, editing, and simulating events, making it possible to connect task models to system models.

Tasks representation in HAMSTERS can be of several types as illustrated in Figure 2. *Abstract task* is a task that involves sub-tasks of different types. *System task* is a task performed only by the system. *User task* is a generic task describing a user activity. It can be specialized as a *motor task* (e.g. a physical activity), a *cognitive task* (e.g. decision making, analysis), or *perceptive task* (e.g. perception of alert). Finally, *interactive task* represents an interaction between the user and the system; it can be refined into *input task* when the users provide input to the system, *output task* when the system provides an output to the user and *input/output task* which is a mix of both but performed in an atomic way.

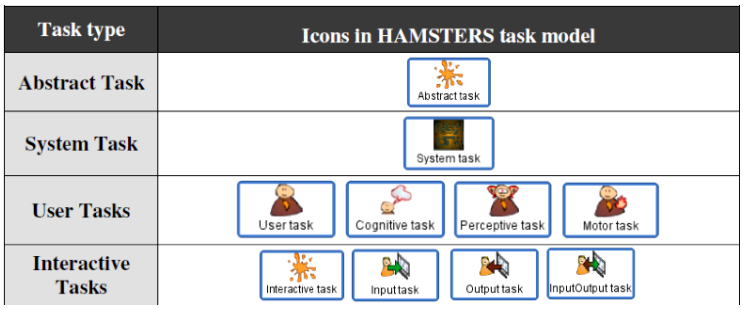

Fig. 2. Task types in HAMSTERS.

Tasks can also have properties. Tasks may be *optional*, *iterative* or both *optional and iterative*. In addition, minimum and maximum execution time can also be set for tasks, and particularly for iterative tasks, it can also be set the information flows. Temporal relationships between tasks are represented by means of operators. The operator *"Enable"* (>>), for example, describes that the tasks T1 and T2 occur sequentially, one after the other. Other operators such as *"Concurrent"* (|||), *"Choice"* ([]), and *"Order independent"* (|=|) describe respectively that tasks can be held simultaneously, the choice of one implies that the other will be disabled, or that the user can choose whether he will perform one or another task first.

The use of operators to link tasks in the model allows extracting the possible scenarios to be performed in the system. This is done by following the multiple achievable paths in the model, with each combination of them generating an executable scenario. HAMSTERS tool supports innately the extraction of scenarios from task models, by simulating their execution and extracting the possible achievable paths. By extracting all the possible scenarios that could be performed in the model, we have a big picture about what (in terms of tasks) can be done with the system. HAMSTERS also provides detailed means for describing data that is required and manipulated in order to accomplish tasks. By using the HAMSTERS' simulation mode, we can set test data at runtime when performing an input task that points to an object in the model.

2.3 An Ontology for Supporting Automated Assessment of GUI-related artifacts

Our approach for supporting automated assessment is based on our previous works which explore an ontology to describe common behaviors with a standard vocabulary for writing BDD stories [35], [38]. The main benefit of this strategy is that BDD stories using this common vocabulary can support specification and execution of automated test scenarios on GUI-related artifacts. The ontology covers concepts related to presentation and behavior of interactive components used in web and mobile applications. It also models concepts describing the structure of BDD stories, tasks, scenarios, and GUIs.

The dialog part of a GUI is described by means of concepts borrowed from abstract state machines. The BDD scenario meant to be run in a given GUI is represented as a transition. States are used to represent the original and resulting GUIs after a transition occur. Scenarios in the transition state always have at least one or more conditions (represented in scenarios by the "Given" clause), one or more events (represented in scenarios by the "When" clause), and one or more actions (represented in scenarios by the "Then" clause). The presentation part of a GUI is described in the ontology through interaction elements which represent an abstraction of the different widgets commonly used in web and mobile user interfaces.

The interactive behaviors in the ontology describe textually how users are supposed to interact with the system whilst manipulating interaction graphical elements of the user interface. An example of behavior specification is illustrated by Figure 3. The specification of behaviors encompasses when the interaction can be performed (using "Given", "When" and/or "Then" clauses), and which interaction elements (i.e. CheckBoxes, TextFields, Buttons, etc.) can be affected. Altogether, interactive behaviors and interaction elements are used to implement the test of the expected system behavior. In the example of Figure 3, the behavior "I choose '<value>' referring to '<field>'" has two parameters: "<value>" and "<field>". The first parameter is associated to data, whilst the second parameter refers to the name given to one of the interaction elements supported by this behavior, which in this example are Radio Buttons, CheckBoxes, Calendars and Links. In green is highlighted the interactive behavior itself (in this case, named in the ontology as *chooseReferringTo*) which will be used for assessing the task name after applying a formatting rule described in Section 5.1.

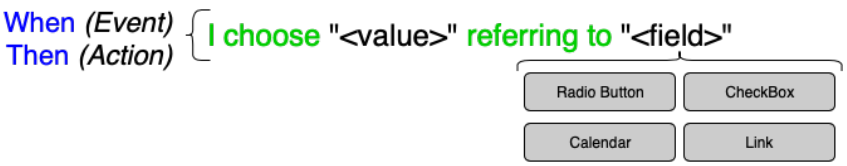

Fig. 3. Structure of an interactive behavior as specified in the ontology.

The ontological model describes only behaviors that report steps performing actions directly on the user interface through interaction elements. This is a powerful resource because it allows keeping the ontological model domain-free, which means it is not subject to particular business characteristics in the BDD stories, promoting the reuse of steps in multiple scenarios. Thus, steps can be easily reused to build different behaviors for different scenarios in different business domains. When representing the various interaction elements that can attend a given behavior, the ontology also allows extending multiple design solutions for the GUI while still keeping the consistency of the interaction. This kind of flexibility leaves the designer free for choosing the best solutions in a given time of the project, without modifying the behavior specified for the system. The current version of the ontology covers more than 60 interactive behaviors and almost 40 interaction elements for both web and mobile user interfaces. A full list of interactive behaviors and interaction elements covered by the ontology can be found in Silva et al. [38].

3 RELATED WORK

Since the early 90's, diverse approaches for improving quality have been studied mainly in the context of viewpoints, both for assessment of requirements acquisition [17], [18] as well as in the context of specification consistency [28]. Viewpoints Oriented Requirements Definition (VORD) was proposed by Kotonya and Somerville [17] as a method to tackle requirements engineering from a viewpoint level. Interactive systems, whose operations involve a degree of user interaction, have

a serious problem in identifying and ensuring that all the clients' needs are recognized in a valid way. The VORD method is useful in detecting these user needs, and also identifying the services that a user expects from the system by providing a structured method for collecting, documenting, analyzing, and specifying viewpoints and their requirements [31]. These methods, however, were targeted to improve the quality and ensure some level of consistency within the requirements specification itself and not between the requirements and other software development artifacts as proposed by our approach.

Even with these and other efforts, artifacts other than final versions of user interfaces are not commonly tested. A common argument is that they cannot be "executed" in order to be tested. Artifacts like task models are usually only inspected manually in an attempt to verify its adequacy [44]. Inspections can be of different types including formal technical reviews, walkthroughs, peer desk check, informal ad-hoc feedback, and so on [45]. When evaluation of the user requirements representation on such artifacts is considered, requirements traceability techniques are employed as a way to trace such requirements along their multiple versions (horizontal traceability) or along their representation in another artifacts (vertical traceability) [12].

Like in our approach which extracts scenarios from task models, other authors [40], [2], [3] have studied means of manipulating task models for obtaining test scenarios. Since a task model describes the whole set of tasks a user can perform in the system, besides providing the set of multiple paths that users are able to follow to accomplish such a task, test cases are obtained by going through these multiple paths, gathering a different execution scenario for each possible path. Therefore, all notations and tools for task modeling provide some kind of mechanism for extracting the set of possible scenarios by simulating a model execution. However, unlike these authors which exploit these multiple paths extracted from task models to generate test scenarios, we use these paths in our approach to allow a direct comparison with the scenarios specified in the BDD stories (which, by the way, already include test scenarios) and then to point out the inconsistencies between them.

There is also an intrinsic relationship between task modeling and user interface design. Wolff et al. [52] proposes to link GUI specifications to abstract dialogue models. Specifications are linked to task models describing behavioral characteristics. Prototypes of interactive systems are refined and interactively generated using a GUI editor. The design cycle goes from task model to abstract user interfaces and finally to a concrete user interface. It is an interesting approach to have a mechanism to control changes in interface elements according to the task to which they are associated in the task models. A similar work is proposed by Luyten et al. [22]. The authors present an algorithm to partially extract the dialog model from the task model. A State Transition Network is proposed to formalize the activity chain which can be partially extracted out of the task specification. The approach allows to verify whether the task and dialog model are consistent. This approach is also useful in automatic user interface generation where several different dialogs are involved. These approaches, however, are oriented to the generation of prototypes and/or dialog models from specifications instead of assessing and verifying the consistency of these models and prototypes with behavior-based user requirements.

Martinie et al. [24], followed by Campos et al. [4], propose a tool-supported framework and a model-based testing approach to support linking task models to an existing, executable, and interactive application. The framework allows developers to define a systematic correspondence between the user interface elements and user tasks. The problem with this approach is that it only covers the interaction of task models with a concrete fully-functional user interface, not covering the assessment of task models concerning user requirements specifications. Another problem is that it requires too much intervention of developers to prepare the source code to support the

integration, making it difficult to be adopted in applications that cannot receive interventions at the code level.

As far as a common vocabulary is a concern (such as defined in the supporting ontology of our approach), the W3C published a glossary of recurrent terms for presentation components called MBUI (Model-based User Interface) [32] on which our work was based. For the dialog component, SWC [50] and SXCML (State Chart XML: State Machine Notation for Control Abstraction) [1] have offered a language based on the state machine concepts. Some authors have also tried to establish a linguistic task modeling for designing user interfaces. Khaddam et al. [16] presented a linguistic task model and notation. The model aims to separate the task and the semantic levels by adopting a well-defined set of task identification criteria. The provided notation enables identification of task input elements based on the task state diagram that is configured on each task. The notation also addressed the dynamic aspect of modeling by introducing dynamic tasks and pumping tasks. This work differs from ours by being focused on the quality of the task specification itself whilst our approach is focused on assessing whether the task specification is consistent with the user requirements specification provided by the BDD stories.

Finally, BDD is nowadays one of the primary software development methods for specifying automated natural language user requirements. Efforts to specify requirements in natural language are not recent though. Language Extended Lexicon (LEL) [19], for example, has been studied since the 90's. The authors propose a lexical analysis of requirements descriptions in order to integrate scenarios into a requirements baseline, making possible tracing their evolution. They were followed by other attempts to identify test cases from requirements specified in natural language [42], [11].

BDD stories have been also evaluated [20] and the characteristics analyzed and studied [43], [13] by several authors. In different contexts from those investigated by our approach, studies have also been conducted to explore the use of BDD as part of empirical analysis of acceptance test-driven development [26]. Other studies have concentrated in the use of automated acceptance testing to support BDD traceability [21], or in analyzing its compatibility with business modeling [8], [9], enterprise modeling [47], [48], [46] and with BPMN [23].

4 THE PROPOSED APPROACH

4.1 Strategy for Assessing Task Models

Both task models and BDD stories support the description of multiple sequences of tasks or actions to be performed by the user/system by means of scenarios. This fact allows us to compare these corresponding sequences in the artifacts and identify whether a given sequence in an artifact can be seen as conflicting with an equivalent sequence in another one. This kind of assessment is illustrated in Figure 4.

In the example illustrated in Figure 4, we can notice the steps A-G from different scenarios of different stories being assessed against a task model and a scenario extracted from it. Therein, the step A at the first position of the first US scenario finds a corresponding task A in the task model once this task could be extracted at the same first position in the TM scenario. In the sequence, the step B at the second position of the first US scenario finds a corresponding task B in the task model which drives the scenario to a specific path in the task model. This task B could be extracted at the same second position in the TM scenario. The other steps and tasks can be analyzed in a similar manner. Notice that the path represented by the sequence of tasks A-B-D-F-G in the task model is only one of the possible combinations of tasks giving origin to the respective TM extracted scenarios presented in the figure.

Therefore, our strategy for assessment is to analyze both the match between step and task names and their respective position in both scenarios. As the steps in BDD scenarios usually contain more

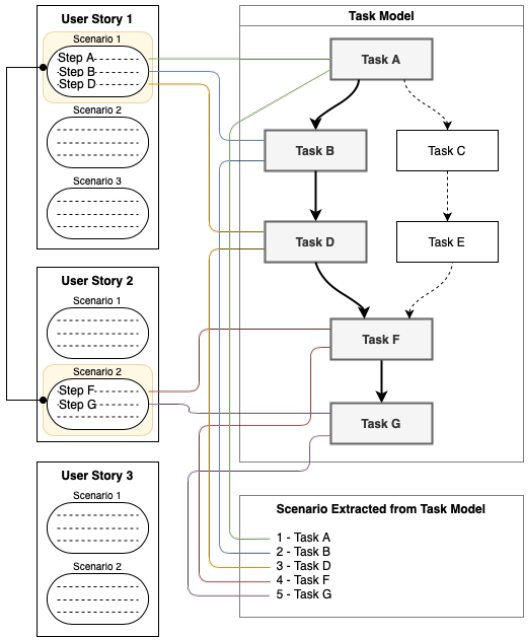

Fig. 4. Task Models Assessment from BDD Stories and Scenarios.

information than it is present in the task names, we firstly apply a formatting rule to the step name before comparing it to a task name. This process is described in detail in Section 5.1. 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. In this case, an inconsistency would be identified if, for example, the step D (which is at the third position in the BDD scenario) found its corresponding task D at the fourth position in the TM scenario. Thus, in summary, we consider there is an inconsistency when:

i We do not find a corresponding task in the TM scenario to match the formatted 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.

The complete automated assessment process to identify these inconsistencies in the TM scenarios is presented in Section 5.

4.2 Preconditions for Assessment

Task models can be designed through a diverse set of notations and tools. For being assessed under our approach though, they need to comply with two premises:

i Allow extraction of scenarios, and
ii Export source files of both reference model and extracted scenarios in a markup language.

The assessment of task models can be performed by either a static or a co-execution approach. The static assessment implies analyzing the source files of the model without actually simulating the execution of the model, whilst co-execution approaches execute it step-by-step providing a visual feedback of the tasks that are being under execution at a given time. Our strategy for testing

performs a static assessment of the source files by means of a syntactic and semantic analysis of the targeted source files. An advantage of this strategy is that, unlike co-execution approaches where artifacts under testing must be prepared for assessment by annotating or modifying their source files, in our approach we have no need to intervene in the source code of the target artifacts, i.e. artifacts do not need to be prepared for testing by designers, so task models and requirements specifications can be assessed in their original state.

In this article, we make use of task models modeled by HAMSTERS once the notation and tool fit our two premises stated above. HAMSTERS exports its task models and extracted scenarios using the XML standard, a well-adopted markup language, so recognized by our approach. The task modeling and the extraction of scenarios that will be presented hereafter has been made by using the HAMSTERS tool, whilst the implementation of the assessment has been made by using the respective XML source files produced by the HAMSTERS tool for each model.

Figure 5 shows elements from the HAMSTERS meta-model (left-side of the figure) and how they relate to the elements from the BDD story meta-model (right-side of the figure). As the tasks are analyzed only after being extracted from the task models for a given TM scenario, the comparison between elements from the meta-models is made only between the *Task* element (from the HAMSTERS meta-model) and the *Step* element (from the BDD story meta-model). After the step is formatted following the rule presented in Section 5.1, the corresponding behavior should match one of the interactive behaviors described in the ontology.

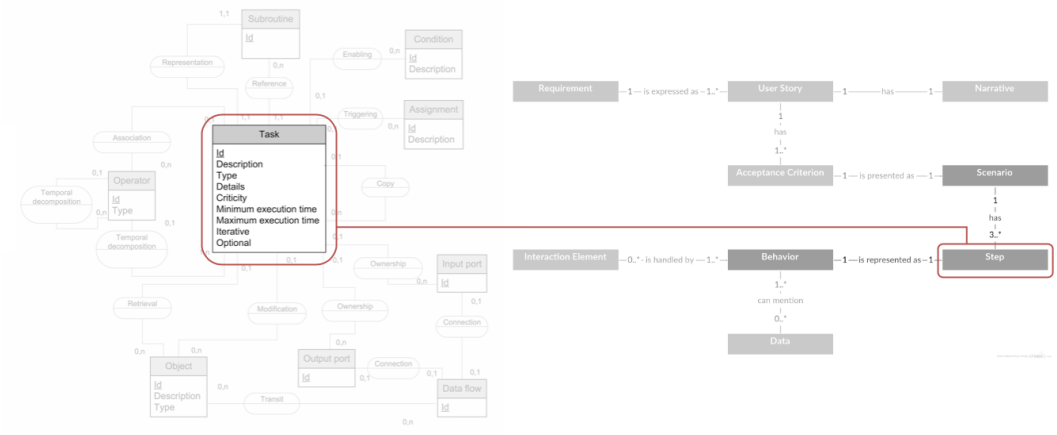

Fig. 5. Relationship between the HAMSTERS meta-model and the BDD story meta-model.

BDD stories and task models must also be at the same abstraction level in order to be assessed, i.e. at the interaction level. At the beginning of a software development project, a single and more comprehensive model can initially be used to represent a higher abstraction level, but at some point, it should be refined to the interaction level in order to be assessed by our approach. Hence, to guarantee the validity of the input and in order to ensure the BDD stories and the scenarios extracted from task models are actually comparable, the designers must, first of all, ensure that both artifacts have the same granularity and level of detail.

4.3 Steps for Performing the Approach

Depending on the project phase, our approach can be applied in two ways. The first one is applied when the project is running, and task models have already been designed. In such a case, our

approach can be used to assess such artifacts, indicating where they are not in accordance with the specified requirements. The second one refers to a project in the beginning, where no task models have been designed yet. In this case, by using the proposed set of interactive behaviors from the ontology, they can be modeled in a consistent way from the beginning.

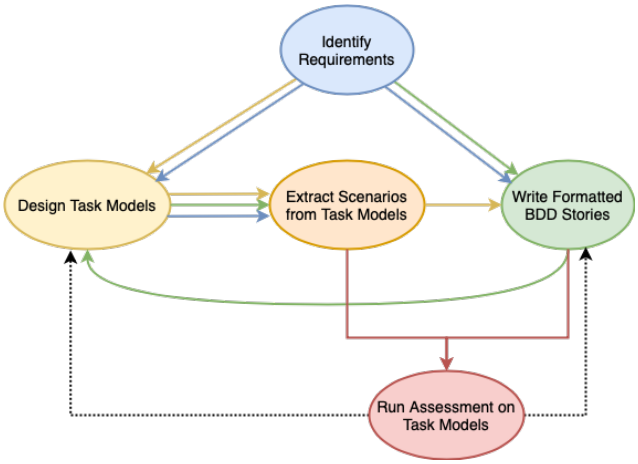

Fig. 6. The graph of options for performing our approach (colors are used to visually identify the different paths).

Figure 6 illustrates the resultant graph of options considered. The colored lines indicate the possible paths to be taken in the workflow. The yellow path indicates the design of task models before writing formatted BDD stories. The green path indicates the opposite, while the blue path indicates both activities in parallel. Notice that regardless the path chosen, the extraction of scenarios from task models (which may be a manual or an automated process depending on the tool used) is only possible after having designed the task models (a manual process), and the identification of requirements (a manual process as well) is a precondition for all the other activities. Finally, to run the assessment on the task models (a fully automated process), it is required to have extracted scenarios from them and written the formatted BDD stories. After running the assessment, the inconsistencies identified can be fixed by redesigning either the task models or the BDD stories (black dotted lines). The approach is intended to benefit requirements engineers and interaction designers in the first place; thus, they are the main stakeholders involved who are responsible to gather and writing the BDD stories besides modeling the task models.

4.4 Challenges associated with the extraction of scenarios

Task models can be tricky to manipulate once the number of possible scenarios can scale exponentially due to the complexity of the model. Different operators, the presence of optional tasks, the number of times an interactive task can be executed, etc. make the extraction of scenarios for testing a very complex activity. Campos et al. [3] illustrate this problem and propose a catalog of strategies for modifying the models in order to manage the complexity of the resulting set of extracted scenarios. An easy-to-see consequence of such kind of strategy is that models are not fully manipulated, i.e. the reference model for extracting test scenarios is a simplified instance (a subset) of the original model. Consequently, several nuances of modeling (such as the use of multiple operators, non-interactive tasks, etc.), which allow task models being a rich representation

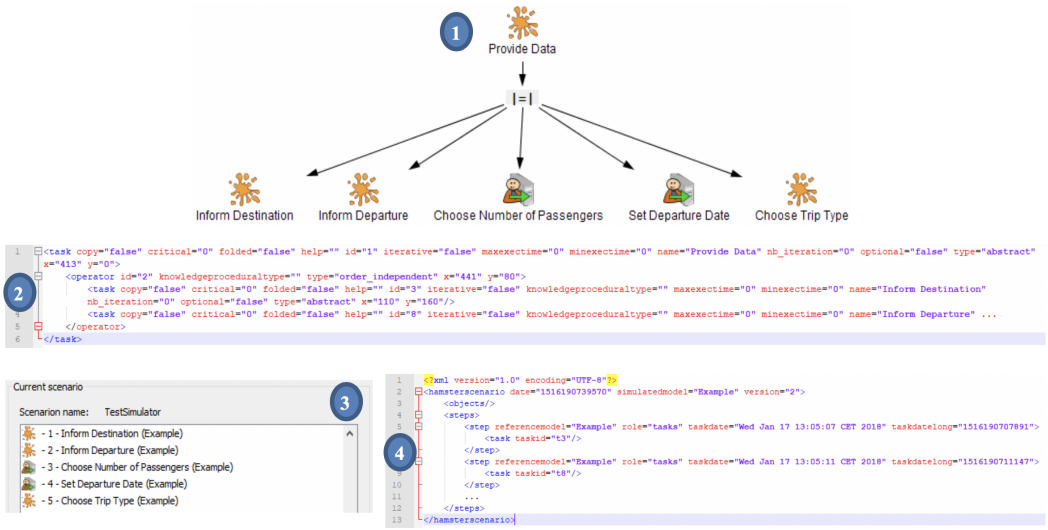

Fig. 7. Task model (1), extracted scenario (3), and their respective source files (2 and 4).

of human activities when interacting with the system, are lost and cannot be verified or even taken into account when obtaining scenarios.

In order to exemplify this problem, Figure 7 takes an example of our current approach for extracting scenarios from task models. The model presents a short extract of some tasks (1 in Figure 7) involved in the process of booking flight tickets through a generic flight booking system. Therein, an abstract task named "*Provide Data*" generalizes a sequence of 5 tasks that can be performed in any order. This attribute is signalized by the operator "*Order Independent*" (|=|) placed between "*Provide Date*" and the other 5 tasks. Thus, one of the possible scenarios that could be extracted from this model is presented further (3 in Figure 7). Therein, tasks are performed in the order they are visually presented in the model, i.e. first the user informs a destination and a departure, then he/she chooses the number of passengers, sets the departure date, and finally chooses his/her trip type.

Notice that the XML source file of the extracted scenario (4 in Figure 7) is just a sequential description of tasks in the model that have been settled for execution. The file brings for each task only a reference for its ID (it does not even bring the name of the task), the source task model, and the date/time of execution. The XML source file of the task model itself (2 in Figure 7) is, on the other hand, a richer description of task modeling elements, including tasks of several types, operators, constraints related to the number of iterations each task supports, tasks that are optional, maximum and minimum time of execution, levels of criticality and so on.

Therefore, we can easily realize that the manipulation of XML source files of task models brings us a full range of challenges. For example, as pointed out by Campos et al. [3], the presence of "order independent" operators between subtasks is a major contributor to the state explosion in the state machine generated from a task model. Considering a simple example in Figure 7, although the task model has only five subtasks following the abstract task "*Provide Data*", the resultant number of possible combinations of tasks (resulting in scenarios) is equal to 120. This happens because we must consider all the permutations of the five tasks' execution. By following all the leaves in a task model with multiple operators, we notice that the number of possible scenarios for extraction gets exponential in function of the types of these operators. That is the reason by which authors

working with task model exploitation for generating scenarios or test cases usually control such an extraction, in order to reduce the complexity and the resultant number of combinations. This becomes especially challenging if task models specify collaborative activities with several instances of the same role.

Like other authors, we have also followed a strategy based on the extraction of scenarios from task models for obtaining test scenarios to check the quality of such models, but unlike other approaches, we have not controlled such an extraction, which allowed us to keep important aspects of the interaction.

5 IMPLEMENTATION

5.1 Formatting Rule for Task Names

As task models are designed to support the multiple paths that users may accomplish to perform their tasks, assessing such models in a scenario-based approach involves initially extracting the possible scenarios that are supposed to be tested at a given time. It means that after modeling, designers should define which branches of the model will be tested. The equivalence of steps in BDD stories and tasks in scenarios extracted from task models is supported by our aforementioned ontology [35], [38] which provides the set of interactive behaviors supported. Our testing algorithm retrieves the respective behavior from the ontology and performs a formatting rule in the step name as exemplified in Figure 8 in order to verify whether a behavior described in a step has an equivalent task to model it in the task model.

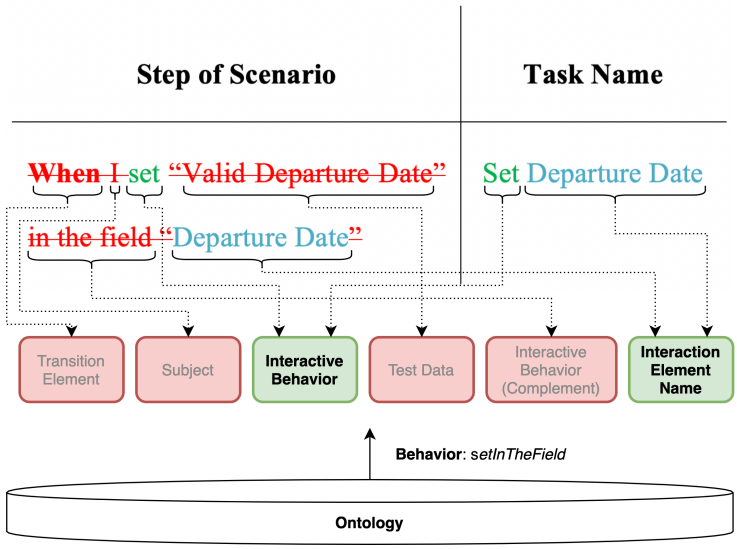

Fig. 8. Formatting rule for assessing steps and tasks.

This rule aims to eliminate unnecessary components of the step name that do not need to be present in the task. 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. Task models already 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 (interactive behavior) that will be performed by the user, so it begins to name the task in the task model. The value "Valid Departure Date" indicates the test data domain that

Table 1. Task name components construction.

Component	Description	Usage for naming tasks
When (Transition Element)	Refers to a transition element of the state machine.	**Not used** because these elements are not addressed in task names.
I (Subject)	Signalizes it is the user who performs the task.	**Not used** because other task model elements already encompass the definition of user role.
set (Interactive Behavior)	Indicates the action (interactive behavior) that will be performed by the user.	**Used** for beginning to name the task in the task model.
"Valid Departure Date" (Test Data)	Indicates a data domain (or the actual test data) that will be used to perform and simulate the task execution.	**Not used** because test data are provided only at runtime when simulating the task model and are not part of the task name.
in the field (Interactive Behavior Complement)	Signalizes that an interaction component (a "field" in this case) will be called.	**Not used** because it is just a complement of the interactive behavior name and usually does not compose the task name.
"Departure Date" (Interaction Element Name)	Indicates the name of the interaction element that will be affected by this task.	**Used** for composing the final name of the task in the task model.

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 and it is provided only at runtime when simulating the execution of the task model. The interactive behavior complement "in the field" just signalizes that an interaction element (a "field" in this case) will be referenced in the sequence. This component is optional and it is not present in all interactive behaviors described in the ontology, so these complements are just dismissed when found. 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 in the task model. Table 1 below summarizes the use of such components for forming the task names to be searched from the step names.

Table 2 provides some examples of task names to be assessed after applying the formatting rule discussed above. Notice, for example, that the step *And I inform "Departure City" and choose "Departure Airport" in the field "Departure"*, which actually encompasses two actions to be performed (*inform "Departure City"* and *choose "Departure Airport"*), is broken into two task names: "Inform Departure" and "Choose Departure". Notice also that for steps described in the passive voice (which is usual when describing outcomes to be verified in the clause "Then"), the resultant task name to be assessed is described in the active voice from the system point of view ("Display List of Available Flights", for example). This kind of task is usually described as an output task in the task model.

5.2 Pre-formatting HAMSTERS Source Files

The automated assessment of task models is made by parsing the resultant XML source files of the extracted TM scenarios produced by the HAMSTERS tool. To do so, we have implemented an integrated algorithm in Java using JDOM and JUnit for parsing and testing the BDD stories against these artifacts.

Table 2. Examples of task names to be assessed after applying the formatting rule

Interactive behavior from the ontology	Step in the BDD scenario	Task name to be assessed
goTo	Given I go to "Find Flights"	Go to Find Flights
chooseReferringTo	When I choose "One way" referring to "Trip Type"	Choose Trip Type
informAndChooseInTheField	And I inform "Departure City" and choose "Departure Airport" in the field "Departure"	Inform Departure / Choose Departure
chooseTheOptionOfValueInTheField	And I choose the option of value "2" in the field "Number of Passengers"	Choose Number of Passengers
setInTheField	And I set "Valid Departure Date" in the field "Departure Date"	Set Departure Date
submit	And I submit "Search"	Submit Search
willBeDisplayed	Then will be displayed "List of Available Flights"	Display List of Available Flights

The first step for assessing the set of scenarios extracted from task models is to preformat their XML files. As each task model notation and tool has its own way to implement and export scenarios and models, and there is no such a standard for that, each notation would demand a different preformatting to be tested by our approach. We have implemented a solution for HAMSTERS in its current version (v4.0), but we have designed a flexible and open architecture where other notations could benefit from our approach by just implementing a new preformatting Java class in accordance with their own patterns to implement scenarios and models.

As mentioned before, HAMSTERS tool exports scenarios with only a reference to the task ID and the object ID that compose the flow. As such, we have to prepare the files for testing. So, before starting the assessment, we edit each XML file of the TM scenario to add:

- The name of the task referenced by each task ID.
- The information about the optionality of each referenced task.
- The object value associated with each task, if it has been provided during the task execution.

All the information is recovered from the reference task model source file that actually contains the whole set of information about each task that has been modeled. Figure 9 illustrates an extract of the original (left side) XML scenario file, and the resultant (right side) XML scenario file after the process of preformatting.

Besides preformatting the XML files of the extracted scenarios, our algorithm also adds, for each scenario, an equivalent scenario without the optional tasks. This is made due to a limitation in the current version of the HAMSTERS tool that does not allow to extract scenarios without the optional tasks. The tool necessarily includes both optional and non-optional tasks present in the model during the process of extracting scenarios. Thus, in order to obtain scenarios without the optional tasks, we algorithmically generate new scenarios eliminating all the tasks signalized as optional in the set of scenarios extracted from HAMSTERS. Such new scenarios are named as "No Optional" followed by the original name of the scenario extracted from HAMSTERS. As a result, for each scenario extracted from HAMSTERS (necessarily including all optional tasks), we generate an additional similar scenario, but without all the optional tasks.

```
…
<step referencemodel="Inform a Flight Leg"
role="subroutines" taskdate="Thu Apr 19 15:01:29 CEST
2018" taskdatelong="1524142889116">
  <task taskid="t13"/>
</step>
<step referencemodel="Search Flights" role="tasks"
taskdate="Thu Apr 19 15:01:40 CEST 2018"
taskdatelong="1524142900016">
  <task taskid="t23">
    <stepObject objectID="6"/>
  </task>
</step>
…
```

```
…
<step referencemodel="Inform a Flight Leg"
role="subroutines" taskdate="Thu Apr 19 15:01:29 CEST
2018" taskdatelong="1524142889116">
  <task taskid="t13" taskname="Set Departure Time Frame"
optional="true" />
</step>
<step referencemodel="Search Flights" role="tasks"
taskdate="Thu Apr 19 15:01:40 CEST 2018"
taskdatelong="1524142900016">
  <task taskid="t23" taskname="Set Arrival Date"
optional="false">
    <stepObject objectID="6" objectContent="Lun, Déc 10,
2018" />
  </task>
</step>
…
```

Fig. 9. Extract of an original (left side) and a resultant (right side) scenario XML files after the process of preformatting.

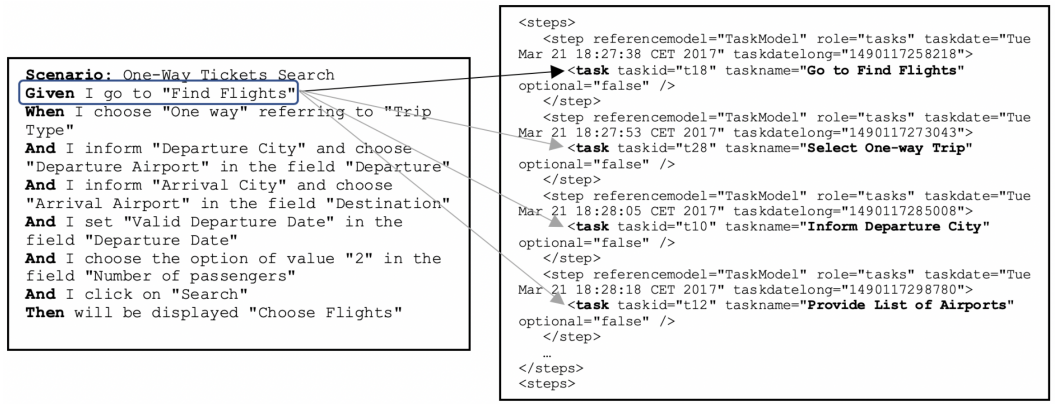

Fig. 10. Checking consistency of tasks between a BDD scenario and scenarios extracted from task models.

5.3 Performing the Assessment

The assessment of BDD scenarios and scenarios extracted from task models consists of verifying, for each step in the BDD scenario, if there are one or more corresponding tasks for such a step in the XML source files of the scenarios extracted from the task models. Corresponding tasks are searched after applying the formatting rule described in Section 5.1 to the name of the step in the BDD scenario. To do so, our algorithm illustrated below fixes a step in the BDD scenario (*"Given I go to 'Find Flights'"* for example) to be verified in the task model, following the mapping presented in Figure 10 (*"Go to 'Find Flights'"* in the example). Then we parse each task of each scenario in the XML source file looking for one or more correspondences to that step. If matches are found, then a list of matches is created, keeping the position in each scenario-task where the match has been found. The algorithm 1 presented below implements such a strategy.

The results of testing are shown in a log indicating, for each step of the BDD scenario, if and where a given step has found an equivalent task in the XML file analyzed, and once it carries an object value associated, which value it is. As scenarios in BDD stories and scenarios in task models may be ordered differently, the algorithm checks the whole set of XML files to ensure we are looking for all the instances of the searched task. Due to that, if there are several XML files of scenarios, the results in the log will show where a corresponding task has been found in each one

```
Result: ListOfMatches
for each step from the BDD scenarios do
    taskToFind ← corresponding task from the ontology ;
    for each task from each XML source file do
        if the attribute taskname is equal to taskToFind then
            ListOfMatches ← position(scenario,task);
        end
    end
end
```

Algorithm 1: Testing algorithm for assessing scenarios extracted from task models.

of them. In summary, the log of results shows, for each step of the BDD scenario, the results of searching in each TM scenario file ("*.scen*"). Each line of results brings then:

- the name of the scenario in which the search has been carried out,
- the task name that has been searched for,
- the position in which the task has been found (if so), otherwise is shown the message "Task not found!", and
- the object value associated with each task (if any), otherwise is shown the message "No Value".

5.4 Tool Support

The testing algorithm we have just described above has been implemented in Java. The project has been structured in two packages. The first one encompasses the classes for implementing the solution. This package contains four classes: *MySteps*, *MyTest*, *MyXML* and *PrepareFiles*. *MySteps* implements the mapping between the interactive behaviors described in the ontology and the assertion that should be made when checking scenarios from task models. *MyXML* implements methods for parsing scenario files extracted from task models in their XML files. *MyTest* is the JUnit class that triggers the set of BDD stories that have been selected for assessment. Finally, *PrepareFiles* is the class in charge of preformatting the scenario source files extracted from task models, as described in Section 5.2.

The second package encompasses the resources demanded for running the assessment. Two folders "stories" and "scenarios" encompass respectively the set of BDD (User) stories text files that have been specified for the project (with a "*.story*" extension), and the current scenario's XML files extracted from task models under assessment before and after the process of preformatting. Finally, a third folder named "task models" keeps the reference XML source files for the task models under assessment. Such files are useful to allow the process of preformatting.

Figure 11 represents the flow of calls we have designed in our algorithm for running a battery of tests on task model scenarios. The flow starts with the class "MyTest.java". First of all, this class instantiates an object from "PrepareFiles.java" (flow 1) in order to trigger the process of preformatting mentioned before. Such a process runs on the package of task model scenarios (flow 2), naming the extracted tasks and adding useful complementary information for testing. For that, the process asks the reference source file ("*.hmst*") of the corresponding task model mentioned by each task in the scenario. After getting the scenario files formatted, "MyTest.java" includes the BDD (User) story (or the set of BDD stories) that will be assessed (flow 3).

Each one of the steps in the BDD (User) story under testing makes a call to the class "MySteps.java" (flow 4) that knows which behaviors are supported by the ontology. Based on the behavior referenced by the step, this class makes a call to the class "MyXML.java" (flow 5) in charge of parsing all the

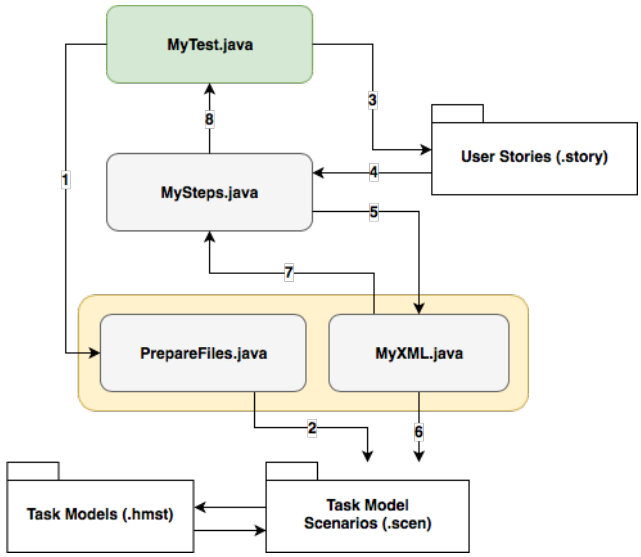

Fig. 11. Flow of calls for running tests on task model scenarios.

set of task model scenarios (flow 6). This parsing aims to check if the behavior addressed by the step is also present in the same position in at least one of the scenarios extracted from the task models. The result of this parsing is then returned to the class "MySteps.java" (flow 7). At this point, based on the algorithm presented in the previous section, a list of all the matches found during the parsing for each step is presented as a result. Finally, the class "MySteps.java" returns the result to the class "MyTest.java" (flow 8) that made the original call.

Notice the independence of the components assigned at the core of the structure represented in Figure 11 (highlighted in yellow). Those components are related to the particularities of test implementation for HAMSTERS task models and scenarios. As mentioned before, "PrepareFiles.java" is in charge of preformatting the extracted scenario files and reading the reference source file of task models, while "MyXML.java" is in charge of parsing the scenario files, searching for the elements under testing. Therefore, we deliver a flexible architecture allowing, in the future, that task models and scenarios modeled by other modeling tools (or even by other versions of HAMSTERS) could also be tested by just implementing new interfaces for this core.

5.5 Setup and Running

Considering the presented architecture, to setup and run a battery of tests, we must:

i place the set of task model scenario files ("*.scen*") that will be assessed in the package "Task Model Scenarios",
ii place the set of task model files ("*.hmst*") that will support the assessment in the package "Task Models",
iii place the set of BDD story files ("*.story*") that will be assessed in the package "User Stories",
iv indicate in the "MyTest" class which BDD story will be assessed, or which folder ("/stories") contains all the BDD stories that will be assessed, and finally
v run the "MyTest" class as a JUnit Test.

Thus, for running the assessment, the "MyTest" class is triggered. This JUnit class specifies exactly which BDD story (or which set of BDD stories) will be run. Results are shown in the console

highlighting, for each step of the BDD scenario, whether and where some corresponding task has been found and which value was associated to it (if any).

6 CASE STUDY

To evaluate our approach, we have conducted a case study with an existing web system for booking business trips in a French research institute. In a previous study [33], domain experts from the department of business trips were invited to produce User Stories (following the "BDD story" template) to describe a feature they considered important to that system. This exercise had as objective evaluating the usage of predefined interactive behaviors, by potential Product Owners (POs), for writing BDD stories. For the present study, the BDD stories produced by the participants have been used to identify examples of relevant scenarios to the domain. Based on that, we refined such stories to get a representative set of user requirements to be assessed on task models of the existing system. To obtain such task models, we have studied the current implementation of the existing system, and by applying a manual reverse engineering [6], we redesigned the targeted models in HAMSTERS. The aim of this software reengineering was to have such artifacts to run our tests and examine which types of inconsistencies our approach would be able to identify.

To perform the assessment, we firstly developed an initial version of BDD stories and their test scenarios to act as our user requirements and acceptance criteria. We then reengineered initial versions of the respective task models. After getting ready a first version of task models, we extracted a representative set of scenarios from them. By following our strategy for assessment, we run this initial version of BDD stories to the initial set of scenarios extracted from task models. Results were then evaluated, and we could observe the type of inconsistency we succeeded identifying. As the strategy we follow for assessing scenarios in both task models and BDD stories parses all the steps of each scenario at once, the first round of results is obtained with a single battery of tests.

6.1 Task Modeling

Task models have been developed for this study by using the HAMSTERS tool. As we have focused on the process of searching and demanding a booking of flights, the four models below have been divided to cover the processes of searching the flights, informing a flight leg (or a new flight leg in case of a multidestination trip), and choosing and confirming (or declining) the selected trip. They are presented as sub-models in the abstraction level required to perform the assessment (i.e. the interaction level), which allows a direct comparison with each one of the BDD stories considered.

Figure 12 presents the task model for searching flights using Travel Planet (the current booking system). All the tasks have been designed to be performed by end users of the system, i.e. researchers from the institute booking their own flights, or the travel department team booking flights on behalf of the researchers. The Search Flight feature encompasses accessing the search flight page (task "Go to Book Flights"), informing at least one flight leg (abstract task "Inform a Flight Leg"), providing flight data for searching (abstract task "Provide Data to Search"), submitting the search (task "Submit Search"), and verifying the resultant list of flights (abstract task "Verify List of Flights"). These four tasks are supposed to be performed exactly in this sequence, so the operator "Enable" has been used.

To inform a flight leg (Figure 13), the user is supposed to inform departure and destination data. Such data include informing a departure and arrival cities and based on a list of available airports in those cities, selecting the ones he/she wants. Both tasks are mandatory and should be performed sequentially, so the operator "Enable" has been used. After selecting the airports of departure and arrival, the user must set in any order the departure date and the departure time frame, being this last one an optional task.

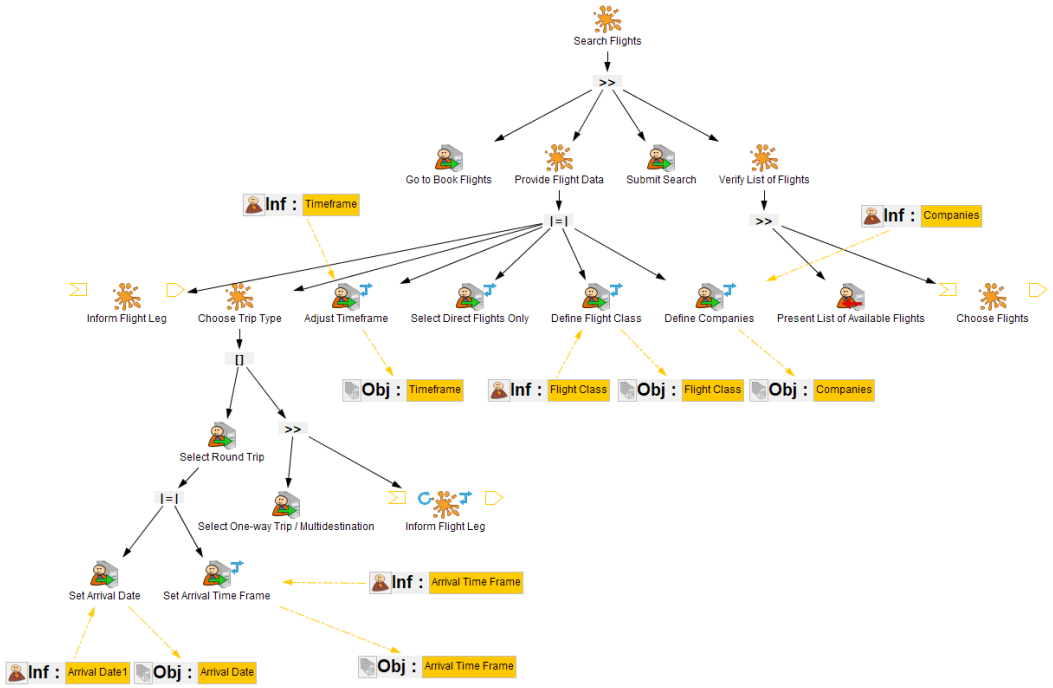

Fig. 12. Task Model for Searching Flights using Travel Planet.

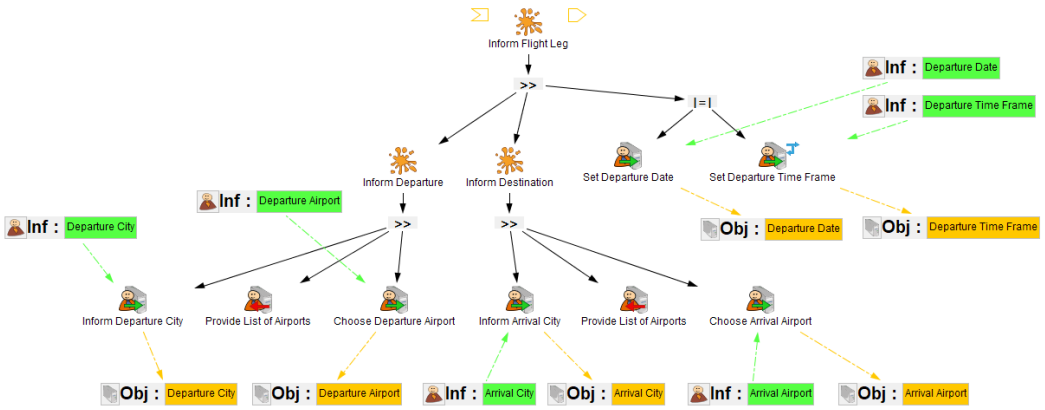

Fig. 13. Task Model for Informing a Flight Leg in Travel Planet.

Going back to providing flight data to search, the user can perform in any order (operator "Order independent") the following tasks: "Choose Trip Type", "Adjust Timeframe", "Select Direct Flights Only", "Define Flight Class", and "Define Companies", being the last four tasks optional. For choosing trip types, the user has three options. If a round-trip is chosen, then a sequence of two order-independent tasks can be performed by the user: "Set Arrival Date" and "Set Arrival Time Frame", being this last one optional. If a multidestination trip is chosen, then the user must inform at least one more flight leg (abstract task "Inform a New Flight Leg"), performing the same interactive

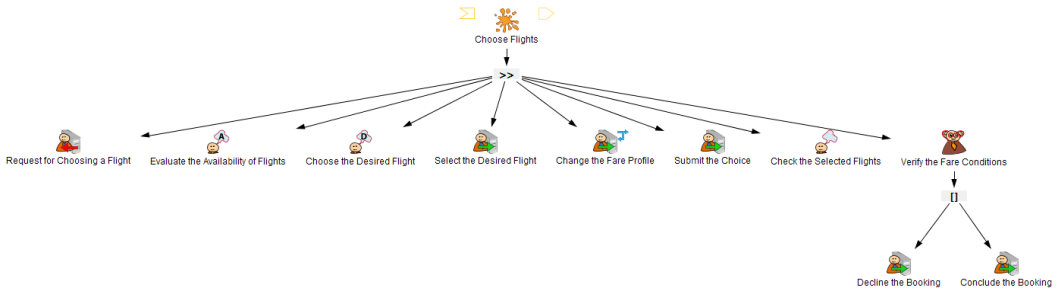

Fig. 14. Task Model for Choosing a Flight in Travel Planet.

tasks from "Inform a Flight Leg". Finally, if a one-way trip is chosen, there is no additional tasks to perform for the abstract task of choosing a trip type. For all the input tasks, notice that the data handling is shown with information being provided as input for the task and objects being the output of these tasks.

After providing the data, the user can submit the search and get, as a result, a list of available flights matching his/her criteria. At this point, the system returns such a list and the user can then pick one of the available flights and confirm or decline his/her booking. For performing such tasks, the abstract task "Choose Flights" has been modeled (Figure 14). To get it done, the user must first evaluate the availability of flights (which is a cognitive analysis task), choose the desired flight (which is a cognitive decision task), and then select the desired flight (which is indeed an interactive input task). Optionally the user can change the fare profile for the flight he/she has chosen, and then submit his/her choice. Lastly, the user checks the selected flights (a cognitive task) and verify the fare conditions (a perceptive task). He/she then finally chooses between decline the booking or conclude it.

6.2 Assessing Task Models

In total, we set up for assessment 3 BDD stories with 15 different scenarios and extracted 10 scenarios to be assessed from the task models presented above. The first TM scenario (illustrated in Figure 16) is intended to book a regular roundtrip (return trip) without including any test data, whilst the second one is intended to the same purpose but providing test data for the objects values during the execution. The third scenario is intended to book a one-way trip, the fourth one to decline a one-way trip, and the fifth one to book a multidestination trip. Each one of these five scenarios is accompanied by similar ones that do not include the optional tasks, which totalizes the 10 scenarios to be assessed. Figure 15 brings an example of a sequence of BDD scenarios for successfully booking a return trip which is supposed to match the scenario "Successful Return Trip – Regular Case" extracted from the task models.

Testing results are shown in a log indicating, for each step of the BDD scenario, (*i*) if and where a given step has found an equivalent task in the XML file analyzed, and (*ii*) once it carries an object value associated, which value it is. Figure 17 (and its corresponding chart in Figure 18) brings the results produced by our algorithm when searching for the position of each one of the tasks that composes the scenario. So, the lines of the table (and the legend of the chart) bring the steps in the BDD scenarios, and the columns (and the series of the chart) bring the XML files of the scenarios extracted from the task models. Zeros (0) in the table indicate that a corresponding task for a given step has not been found in the target file. Values different than zero indicate the position where a corresponding task has been found in the target file. We highlighted in gray at the

```
BDD Story 1: Flight Tickets Search

Narrative:
As a researcher
I want to be able to search air tickets for my business trips, providing destinations and dates
So that I can obtain information about rates and times of the flights.

...

Scenario: Successful Roundtrip Tickets Search With Full Options
Proceed to Login
Reach the Travel Planet Search Page
Given I go to "Flight Search"
When I inform "Toulouse" and choose "Toulouse, Blagnac (TLS)" in the field "Departure"
And I inform "Paris" and choose "Paris, Charles-de-Gaulle (CDG)" in the field "Destination"
When I set "Sam, Déc 1, 2018" in the field "Departure Date"
And I set "08:00" in the field "Departure Time Frame"
When I choose "Round Trip"
And I set "Lun, Déc 10, 2018" in the field "Arrival Date"
When I set "10:00" in the field "Arrival Time Frame"
And I choose the option of value "2" in the field "Number of Passengers"
When I set "6" in the field "Timeframe"
And I select "Direct Flights Only"
When I choose the option of value "Economique" in the field "Flight Class"
And I set "Air France" in the field "Company 1"
When I submit "Search"
Then will be displayed "2. Sélectionner un voyage"

...

BDD Story 2: Select a Suitable Flight

Narrative:
As a researcher
I want to get a list of compatible flights (including their rates and times) in accordance with my search criteria
So that I can select a suitable flight based on my needs.

...

Scenario: Select a Return Flight Searched With Full Options

Successful Roundtrip Tickets Search With Full Options
Given "Availability Page" is displayed
When I click on "No Bag" referring to "Air France 7519"
And I click on "No Bag" referring to "Air France 7522"
When I click on "Book"
Then will be displayed "J'accepte les Conditions d'achat concernant le(s) tarif(s) aérien(s)."

...

BDD Story 3: Confirm Flight Selection

Narrative:
As a researcher
I want to get all the required data to confirm my flights
So that I can check the information, the fare rules and then finalize my booking.

...

Scenario: Confirm a Flight Selection (Full Version)

Select a Return Flight Searched With Full Options
Given "Confirmation Page" is displayed
When I choose "I accept the General Terms and Conditions."
And I click on "Finalize the trip"
Then will be displayed "Votre voyage a été confirmé!"

...
```

Fig. 15. Sequence of BDD scenarios for successfully booking a return trip.

```
Scenario 1: Successful Return Trip - Regular
Case

1 - Go to Book Flights
2 - Inform Departure City
3 - Provide List of Airports
4 - Choose Departure Airport
5 - Inform Arrival City
6 - Provide List of Airports
7 - Choose Arrival Airport
8 - Set Departure Date
9 - Set Departure Time Frame
10 - Set Arrival Date
11 - Set Arrival Time Frame
12 - Choose Number of Passengers
13 - Adjust Timeframe
14 - Select Direct Flights Only
15 - Define Flight Class
16 - Define Companies
17 - Submit Search
18 - Present List of Available Flights
19 - Request for Choosing a Flight
20 - Evaluate the Availability of Flights
21 - Choose the Desired Flight
22 - Select the Desired Flight
23 - Submit the Choice
24 - Check the Selected Flights
25 - Conclude the Booking
```

Fig. 16. The scenario "Successful Return Trip – Regular Case" extracted from the task models.

table which columns bring the most suitable target files where the correspondence with the BDD scenario was supposed to be found. For a fully consistent model, it would be necessary that each step in the BDD scenario has found its corresponding task in the same position in the target file. So, in this case, a straight vertical line of points would be seen in the chart below, indicating that a sequential correspondence for each step was found. In the first tested scenario presented above, such a correspondence was supposed to be found in the target file "No Optional Successful Return Trip - Regular Case" once, theoretically, it represents the same user activities in the task model.

6.2.1 First Scenario. Analyzing the results of this first round of tests, we can notice that most of the steps have not found a corresponding task in the target files, i.e. steps in the BDD scenarios and tasks in the task models are not consistent somehow. The step at the position 1 (*"Given I go to 'Flight Search'"*) has not found a corresponding task in any target file because, in the task model, the equivalent task has been modeled as *"Go to Book Flights"*, so an inconsistency has been found in the name, despite the position is correct. The step at the position 2 (*"When I select 'Round Trip'"*) has not found a corresponding task in any target file due to a different reason. As the interactive task *"Select Round Trip"* in the task model (see Figure 12) has been modeled as a parent task of two leaf tasks (*"Set Arrival Date"* and *"Set Arrival Time Frame"*), when extracting scenarios from such a model, due to a limitation of the HAMSTERS tool, only the leaf tasks are kept, so the extracted scenario does never show the interactive task *"Select Round Trip"* which is a non-leaf task. Particularly in this case, even if the non-leaf tasks were extracted, another inconsistency would also have been found:

Scenario: Confirm a Flight Selection	No Optional Return Trip With Data	No Optional Successful Multidestination Trip - Regular Case	(Copy) No Optional Successful Multidestination Trip - Regular Case	Successful Return Trip - Regular Case	Successful Multidestination Trip - Regular Case	(Copy) Successful Multidestination Trip - Regular Case	No Optional One-Way Trip Declined	Return Trip With Data	No Optional Successful Return Trip - Regular Case	Successful One-Way Trip - Regular Case	One-Way Trip Declined	No Optional Successful One-Way Trip - Regular Case
Given I go to "Flight Search"	0	0		0	0		0	0	0	0	0	0
When I select "Round Trip"	0	0		0	0		0	0	0	0	0	0
And I inform "Toulouse"	0	0		0	0		0	0	0	0	0	0
and choose "Toulouse, Blagnac (TLS)" in the field "Departure"	0	0		0	0		0	0	0	0	0	0
When I inform "Paris"	0	0		0	0		0	0	0	0	0	0
and choose "Paris, Charles-de-Gaulle (CDG)" in the field "Destination"	0	0		0	0		0	0	0	0	0	0
And I set "Sam, Déc 1, 2018" in the field "Departure Date"	8	8	15	8	16	8	8	8	8	8	8	8
When I set "Lun, Déc 10, 2018" in the field "Arrival Date"	9	0		10	0		0	10	9	0	0	0
And I submit "Search"	11	17		17	23		11	17	11	16	16	11
Then will be displayed "2. Sélectionner un voyage"	0	0		0	0		0	0	0	0	0	0
Given "Availability Page" is displayed	0	0		0	0		0	0	0	0	0	0
When I click on "No Bag" referring to "Air France 7519"	0	0		0	0		0	0	0	0	0	0
And I click on "No Bag" referring to "Air France 7522"	0	0		0	0		0	0	0	0	0	0
When I click on "Book"	0	0		0	0		0	0	0	0	0	0
Then will be displayed "J'accepte les Conditions d'achat concernant le(s) tarif(s) aérien(s)."	0	0		0	0		0	0	0	0	0	0
Given "Confirmation Page" is displayed	0	0		0	0		0	0	0	0	0	0
When I choose "J'accepte les Conditions d'achat concernant le(s) tarif(s) aérien(s)."	0	0		0	0		0	0	0	0	0	0
And I click on "Finalize the trip"	0	0		0	0		0	0	0	0	0	0
Then will be displayed "Votre voyage a été confirmé!"	0	0		0	0		0	0	0	0	0	0

Fig. 17. Test results of the scenario "Confirm a Flight Selection" on the task models.

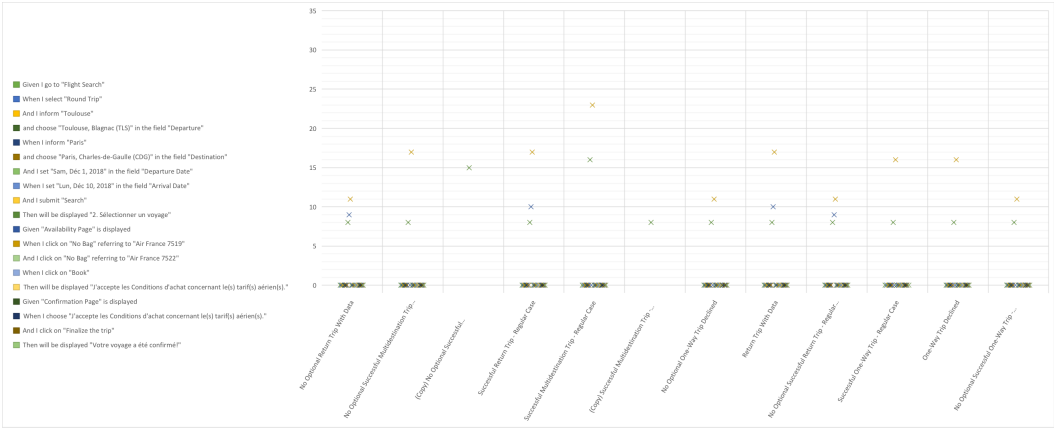

Fig. 18. Results of matching: scenario "Confirm a Flight Selection".

the position in which such a task would appear would not be the second one since it is not among the first user tasks according to the model.

Steps at the positions 3-4 and 5-6 concern respectively the set informing/choosing departure and informing/choosing destination. Such tasks have been modeled (and extracted to scenarios) as the triad "*Inform Departure City* / *Provide List of Airports* / *Choose Departure Airport*" and "*Inform Arrival City* / *Provide List of Airports* / *Choose Arrival Airport*". The intermediate task *"Provide List of Airports"* (that models the output of the system to the user) has not been modeled in the BDD stories, so the step is just composed by the informing and choosing activities. For this reason, such sequence would never find a correspondence in the model, which inevitably would break the forward sequence of tasks in the scenarios. Additionally, another inconsistency that would be identified is that the task model brings tasks named *"Inform Departure (Arrival) City"* and *"Choose Departure (Arrival) Airport"*, while the algorithm would search for tasks named *"Inform Departure (Destination)"* and *"Choose Departure (Destination)"*.

The step at the position 7 (*"And I set 'Sam, Déc 1, 2018' in the field 'Departure Date'"*) has found a corresponding task in all the target files, almost always at the position 8. This one-position gap is due to the absence of the task *"Choose Round Trip"* that was not exported to the scenario as

explained above. Besides that, such a step has found two (instead of one) corresponding tasks in the same file. This happened in the target files *"Successful Multidestination Trip - Regular Case"* and *"No Optional Successful Multidestination Trip - Regular Case"*, exactly the two ones that describe scenarios for a multidestination trip. As in a multidestination trip, the user must inform at least two flight legs, he/she necessarily needs to inform a *"Departure Date"* two times, one for each flight leg. That is the reason the algorithm finds the corresponding task *"Set Departure Date"* two times in these two target files. In the first one, such a task has been found at the positions 8 and 16, and in the second one at the positions 8 and 15. The second occurrence of the task in these files has been marked as "*(Copy)*" in the table of results presented above. Notice that the associated value informed during the extraction of scenario can also be checked with the value specified in the step. The extracted scenario *"Return Trip With Data"* in both versions (with and without optional tasks) brings the associated value *"Sam, Déc 1, 2018"* in the results, that is exactly the same value informed for the corresponding step in the User Story.

The step at the position 8 (*"When I set 'Lun, Déc 10, 2018' in the field 'Arrival Date'"*) has found a corresponding task at the position 9 in the target files *"No Optional Return Trip With Data"* and *"No Optional Successful Return Trip - Regular Case"*, and at the position 10 in the target files *"Successful Return Trip - Regular Case"* and *"Return Trip With Data"*. The task *"Set Arrival Date"* has been found only in those four files because it is only performed in scenarios involving roundtrips (return trips), where an arrival data should be informed. Concerning the position where this task has been found, in the "*no-optional*" files, it has been found at the position 9 because despite the absence of the task *"Choose Round Trip"* (which would bring the task *"Set Arrival Date"* to the position 7), the presence of the two tasks *"Provide List of Airports"* to inform both departure and destination brings the task *"Set Arrival Date"* two positions forward, putting it at the position 9. The position 10 in the target files with optional tasks is due to the presence of the optional task *"Set Departure Time Frame"* right before the task *"Set Arrival Date"*.

The step at the position 9 (*"And I submit 'Search'"*) has found a corresponding task at different positions in all the target files. The task *"Submit Search"* has been found at the position 11 in the "*no-optional*" files (except for the multidestination case that involves informing another flight leg). Looking at the roundtrip case, it highlights an important inconsistency between the scenario presented in the BDD story and those extracted from the task model. Apart from the aforementioned absence of the task *"Choose Round Trip"* and the presence of the two tasks *"Provide List of Airports"* (which would bring the task *"Submit Search"* to the position 10), the fact of being found at the position 11 is due to the presence of a previous task named *"Choose Number of Passengers"* intended to choose the number of passengers that will be included in the booking. This is a mandatory task in the task model but has not been specified as a step in the BDD scenario. It is up to requirements engineers and designers to analyze the models and identify if such a task has been correctly modeled as a mandatory task (so the task model would be correct, and the error would be in the BDD stories), or if it is not the case and such a task should be marked as optional in the task model (so the error would be in the task model and not in the BDD stories).

Steps from the position 10 until 19 have not found a corresponding task in any target file. At the position 10, it was expected the task *"Display 2. Sélectionner un voyage"* and the task model brings the task *"Present List of Available Flights"*. Actually, the task model describes the system task intent which is to present the resulting list of available flights after the search. However, the step in the BDD scenario has specified a given message that would be seen after submitting the search. We can infer that the overall goal of both is the same, but they were specified differently, so there is an inconsistency anyway. At the position 11, it was expected the task *"Display Availability Page"* and the task model brings the task *"Request for Choosing a Flight"*. The system action of requesting the user to choose a flight is performed in the availability page, so both tasks could eventually aim at

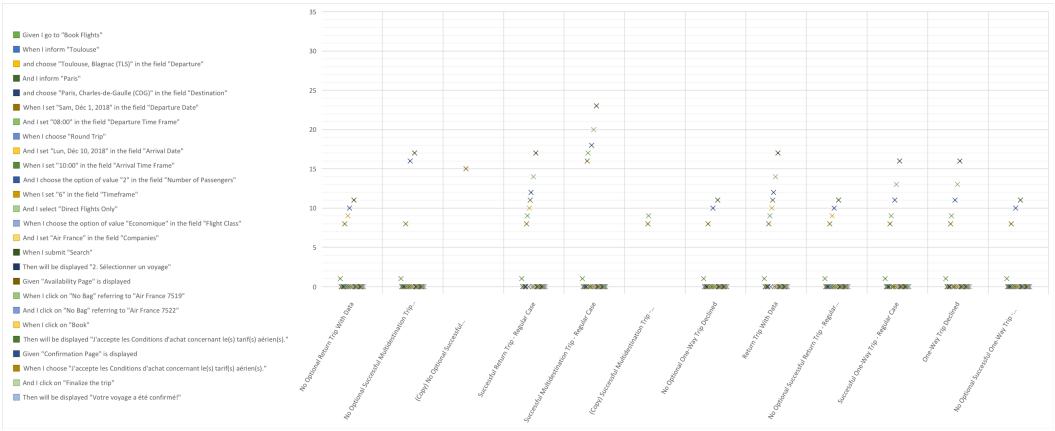

Fig. 19. Results of matching: scenario "Confirm a Flight Selection (Full Version)".

the same purpose, but they are not equivalent once they use different specification strategies. The same occurs with the previous tasks discussed right before.

At the positions 12 and 13, the searched tasks *"Click on No Bag"* and *"Click on No Bag"* would find a correspondence with the task *"Select the Desired Flight"*, but as they specify different behaviors, they cannot be recognized as equivalent. At the position 14, it was expected the task *"Click on Book"* and the task model brings the task *"Submit the Choice"*. Due to the use of different semantic behaviors and the lack of context when analyzing only the tasks individually, it is hard to conclude if both tasks intend actually to model the same behavior.

Steps at the positions 15, 16, 17 and 19 do not have tasks modeling the same behaviors in the task model. The searched task *"Click on Finalize the trip"* at the position 18, just like the ones at the positions from 12 until 14, would find a correspondence with the task *"Conclude the Booking"* extracted from the task model, however they actually specify different behaviors, so they cannot be recognized as equivalent. Notice finally that the tasks *"Evaluate the Availability of Flights"*, *"Choose the Desired Flight"* and *"Check the Selected Flights"*, both of them included in the scenarios extracted from the task models, are cognitive tasks, so they would not be identifiable by the steps anyway. Table 3 summarizes the main reasons of failure discussed above for each step of the BDD scenarios.

6.2.2 Further Scenarios. The sequence of figures 19, 20, 21, and 22 shows the testing results for the remaining scenarios. The second scenario "*Confirm a Flight Selection (Full Version)*" (Figure 19) describes the same roundtrip booking but using all the optional fields. Notice that this scenario brings some fixtures for the inconsistency problems identified with the testing of the previous scenario. For example, the first step has been modified to *"Given I go to 'Book Flights'"* instead of *"Given I go to 'Flight Search'"*, and the step describing the roundtrip selection has been moved forward. Other remarks can be made, notice that as this scenario describes a roundtrip by using the full range of search options, optional steps are never found in the "*no optional*" target files.

Also notice that despite finding a corresponding task in all the target files, we can see that the step *"And I choose the option of value '2' in the field 'Number of Passengers'"* sets the value "2" for the field "Number of Passengers" while in the target file "*Return Trip With Data*" in its both versions (with and without optional tasks), it has been informed the value "1" during the execution. Considering that values specified for test cases are generally representative of a data domain that may point out some failure in the system, it is important to look carefully at such kind of inconsistency in the assessed artifacts.

Table 3. Type of inconsistencies identified in scenarios extracted from task models.

Step	Main reason of failure
1 - Given I go to 'Flight Search'	Task with different name
2 - When I select 'Round Trip'	Task not extracted to the scenario
3 - And I inform 'Toulouse'...	Triple and not double sequence of tasks in the task model
4 - ...and choose 'Toulouse, Blagnac (TLS)' in the field 'Departure'	Triple and not double sequence of tasks in the task model
5 - When I inform 'Paris'...	Triple and not double sequence of tasks in the task model
6 - ...and choose 'Paris, Charles-de-Gaulle (CDG)' in the field 'Destination'	Triple and not double sequence of tasks in the task model
7 - And I set 'Sam, Déc 1, 2018' in the field 'Departure Date'	Wrong position
8 - When I set 'Lun, Déc 10, 2018' in the field 'Arrival Date'	Wrong position
9 - And I submit 'Search'	Inconsistency between modeling and specification
10 - Then will be displayed '2. Sélectionner un voyage'	Different specification strategy
11 - Given 'Availability Page' is displayed	Different specification strategy
12 - When I click on 'No Bag' referring to 'Air France 7519'	Unpaired behaviors
13 - And I click on 'No Bag' referring to 'Air France 7522'	Unpaired behaviors
14 - When I click on 'Book'	Unpaired behaviors
15 - Then will be displayed 'J'accepte les Conditions d'achat concernant le(s) tarif(s) aérien(s).'	Equivalent behavior missing
16 - Given 'Confirmation Page' is displayed	Equivalent behavior missing
17 - When I choose 'J'accepte les Conditions d'achat concernant le(s) tarif(s) aérien(s).'	Equivalent behavior missing
18 - And I click on 'Finalize the trip'	Unpaired behaviors
19 - Then will be displayed 'Votre voyage a été confirmé!'	Equivalent behavior missing

Results of the remaining scenarios, including scenarios to confirm and decline a one-way trip, and confirm a multidestination trip, are shown below in figures 20, 21 and 22. As task models are not designed to model user's errors, scenarios from the BDD stories that test error situations were not assessed with the extracted task model scenarios. As user errors are not part of a user goal, they are usually omitted from tasks descriptions, making this kind of test fail. Means of representing these potential errors on task models is being recently studied [14]. Once it is implemented in the model, tests could run using the same approach to identify this kind of error.

7 DISCUSSION

7.1 Types of Inconsistencies Identified

By summarizing the results presented above, we can categorize the types of inconsistencies found by our approach when assessing the task models. These types were additionally classified by their level of criticality, i.e. the impact of the inconsistency in the system design and the effort that would

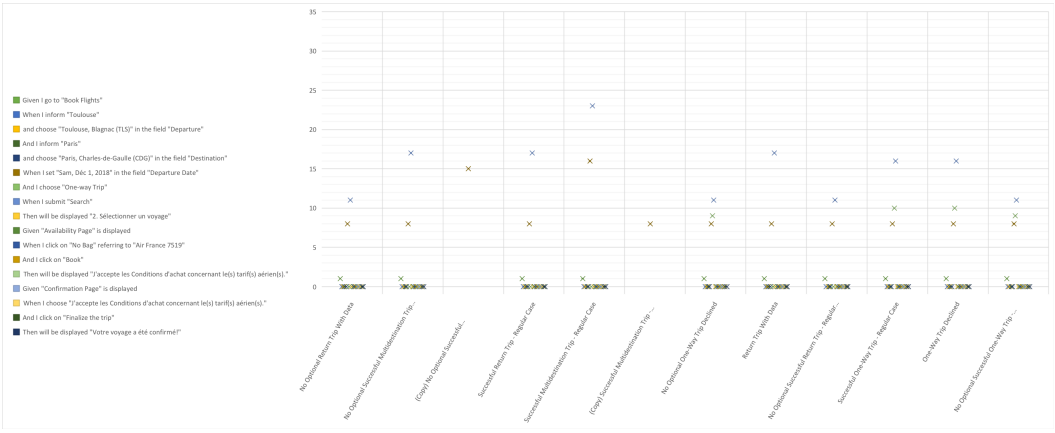

Fig. 20. Results of matching: scenario "Confirm a Flight Selection for a One-Way Trip".

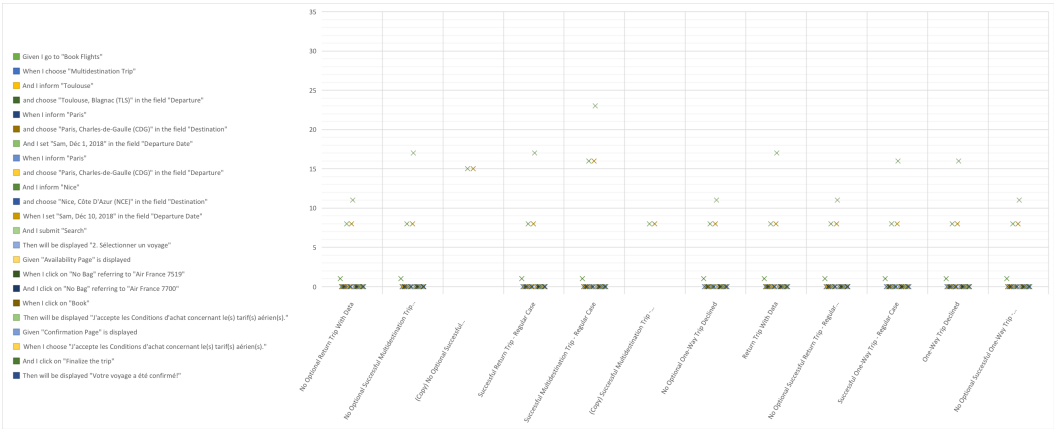

Fig. 21. Results of matching: scenario "Confirm a Flight Selection for a Multidestination Trip".

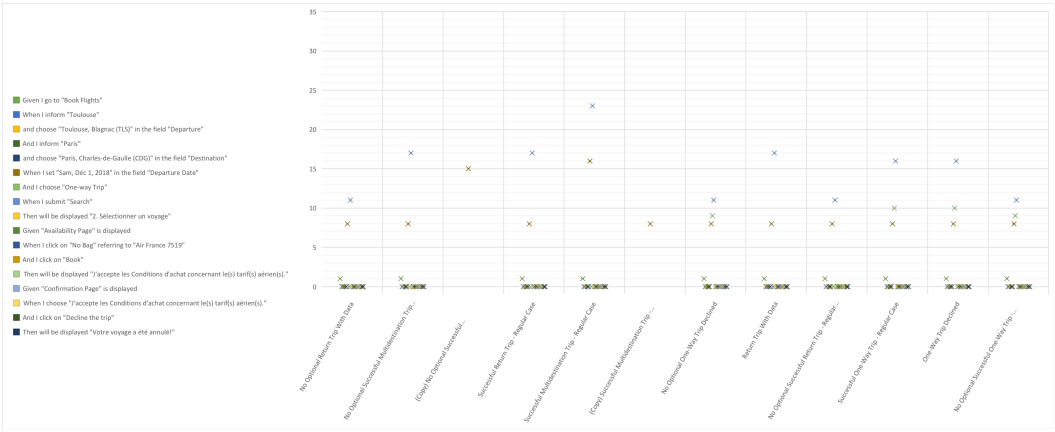

Fig. 22. Results of matching: scenario "Decline a Flight Selection".

be required to fix the problem. These aspects are exemplified and discussed in detail below. We use the symbol (+) to signalize a low-level of criticality, (++) to signalize a medium-level of criticality, and (+++) to signalize a high-level of criticality.

- Task with different names (+)
- Task not extracted to the scenario (++)
- Different number of tasks sequences in the task model (++)
- Wrong position (++)
- Conflict between specification and modeling (+++)
- Different specification strategies (+++)
- Unpaired behaviors (++)
- Equivalent behaviors missing (++)

We succeeded identifying 8 different types of inconsistencies in the assessed scenarios. The most common ones were the "*different number of tasks sequences in the task model*", "*unpaired behaviors*", and "*equivalent behaviors missing*". The first type occurs when there are more tasks in the task model scenario than steps in the BDD scenario to accomplish the same behavior. In the example presented in section 6.2.1, to inform a departure (or a destination) there was a sequence of 3 tasks in the scenario extracted from the task model, while in the step of the BDD scenario, a double action of informing and choosing was enough. For the second type, "*unpaired behaviors*" refers to tasks that would find a correspondence with the steps in the BDD scenarios, but as they actually specify different behaviors (e.g. "*Define <something>*" instead of "*Select <something>*"), they cannot be recognized as such. "*Equivalent behaviors missing*" refers to behaviors that are actually missing in the extracted task model scenario, like steps that are present in the BDD scenario but cannot find corresponding tasks in the task model.

"*Different specification strategies*" comes next as the type of inconsistency incurred from specification of behaviors that could eventually aim at the same purpose, but were specified using different strategies, i.e. requiring to perform (or verify) different actions. An example from this case study is the situation in which a step of a BDD scenario had described a behavior in which the system showed a message introducing a list of available flights, and the task model, a behavior in which the system provided the aforementioned list. Even with the resultant state of the system being the same in this case, the specified behaviors could not be considered equivalents once they use different specification strategies.

Tasks in "*wrong positions*" comes next being the type of error related to tasks that are found in different positions than their equivalent steps in BDD scenarios. As scenarios in different conceptions are being compared when assessing BDD stories and task models, we consider that errors found in the sequence of tasks in the task models (compared with BDD stories) are generally the most sensitive type of error, once it impacts in all other tasks in the sequence. A simple change of task positions at the beginning of a scenario invalidates the whole scenario because all the tasks in the sequence would be in wrong positions. A correction to a simple error like this would include finding the root of the problem, redesign either the step (that would impact the consistency in other artifacts) or the task model (that would imply in extracting new scenarios for testing) and run a complete battery of regression tests again. Considering that there are no other types of inconsistencies in the model, by fixing this issue (either by updating the BDD scenarios to comply with the scenario extracted from the task model or updating the task model to comply with the sequence of steps from the BDD scenarios), both scenarios would become fully consistent.

"*Conflicts between specification and modeling*" refers to tasks modeled in the task model (and consequently exported to its scenarios) that are not present in the requirements specification in the BDD stories. The contrary can occur as well. This kind of inconsistency generally puts

in evidence important conflicts between what is specified in the user requirements and what is effectively modeled in the artifacts. "*Tasks with different names*" and "*Tasks not extracted to the scenario*" complete the list of type of errors encountered during the tests. The first one refers to tasks that are present both in the task model and in the BDD scenario but written with a different name. The second one refers to tasks that are effectively modeled in the task model but, due to the type of operators used or the presence (or not) of other refined tasks after them in the model, causes that, during the extraction process, such tasks are not taken to the extracted scenarios.

By analyzing the variety of inconsistency problems that have been identified in this case study, we can remark that some types of inconsistencies have shown to be more critical than others. While simple inconsistencies like differences in names of tasks and fields are easy to be solved, some other inconsistencies can reveal crucial problems of modeling or important incompatibilities between the requirements specification and its modeling in the artifacts. For example, "*tasks with different names*" would be an easy-to-solve problem, i.e. either the task or the BDD step could be easily renamed to get it sorted out, and this does not heavily affect the design quality and consistency as a whole. This kind of problem could be also easily avoided with, for example, the use of a dictionary. "*Conflicts between specification and modeling*" along with "*different specification strategies*" for task models, on the other hand, compose a more critical group of problems and must be prioritized.

"*Unpaired behaviors*" and "*equivalent behaviors missing*" on task models are inconsistencies directly related to the variety of interactive behaviors modeled in the ontology since the assessment is limited by the set of currently supported behaviors. By supporting new interactive behaviors in the future, the ontology would increase its capacity of recognizing other task descriptions. "*Tasks not extracted to the scenario*", "*different number of task sequences in the task model*", and the presence of tasks in wrong positions are problems that can have the same origin in scenarios extracted from task models, i.e. due to tasks not extracted to scenarios, those ones that have been extracted may be placed in wrong positions which will affect how many sequences of tasks the scenario lists. So, by fixing the root cause, the designer can avoid three kinds of different problems.

7.2 Limitations

7.2.1 Limitations of the Case Study. As limitations of this study, we point out that it has been conducted performing a manual reverse engineering of the existing system currently in production to obtain the respective task models for testing. Therefore, as a manual process, it was expected that inconsistencies would be naturally introduced during the modeling. Indeed, these inconsistencies were identified and that allowed us to evaluate our approach. Nonetheless, if an automated approach of reverse engineering had been used instead, such inconsistencies would probably not have taken place since the task model would be generated already fully consistent with the current system and, by consequence, with the set of BDD stories used to implement it. Future studies should confirm this hypothesis. We also acknowledge a possible modeling bias in the study once both the conduction of the study and the interpretation and analysis of the results have been made by one of the authors. To mitigate that, the results were cross-checked by independent reviewers, experts in interactive systems modeling. They examined both the reengineered task models and the testing results, then they performed a qualitative analysis of the types of inconsistencies identified. The results presented in this article are thus a consolidated and revised version of the testing outcomes.

7.2.2 Limitations of the Approach. As an automated approach, in principle, there is no limitation regarding the size of the model and/or the operators included, nor regarding the complexity of the possible scenarios to be generated. However, the need for extracting scenarios for assessing the task models imposes some limitations. Figure 23 illustrates the flow of activities we have performed so far to obtain scenarios for testing based on the current approaches in the literature. Notice that

the source file of a task model is manipulated only for extracting scenarios (continuous black line at the top). Such a feature is usually included in tool-supported notations for designing task models. After such an extraction, the resultant source files of scenarios are manipulated and formatted to obtain a given scenario for testing (continuous black line at the right).

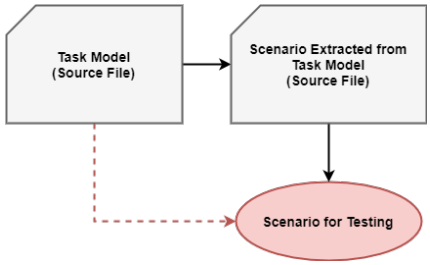

Fig. 23. Flow of activities to get scenarios for testing.

This approach, regardless of keeping important aspects of the interaction during the extraction process, has limitations once it is still fully dependent on the extraction of scenarios, i.e. it does not allow us to manipulate the source file of task models directly. An approach that does not necessarily pass through extracted scenarios (dotted red line at the left) would allow us to manipulate the model with its full capabilities, opening a wide range of opportunities to assess the quality of such models and obtain no-simplified scenarios for testing. This could also solve the problem of not being able to assess non-leaf tasks in the task model. In short, being able to manipulate and check the consistency of task models directly in their source files (instead of passing by the process of extracting scenarios) could represent a crucial step towards a solution that includes a complete and no-simplified assessment strategy. This should be addressed in future works.

8 CONCLUSION

Our approach for assessing task models has the main advantage of signalizing a wide range of inconsistencies that should be fixed to ensure a reliable correspondence between such models and the user requirements specified by stakeholders through BDD stories. Despite the limitations related to the process of extracting scenarios from task models, the strategy we have chosen has many advantages. Especially in environments requiring high availability of tests to be executed continuously throughout multiple iterations, our approach benefits from an instantaneous consistency checking, allowing the analysis of several hundreds of scenario files at the same time.

Unlike our approach (which is based on a static analysis), co-execution approaches have the benefit of allowing simulating the execution of models simultaneously with visual feedback in real-time about the correspondence of entities that are being assessed in each model. However, such approaches demand a high investment to prepare the models before, annotating the source code/files or even modifying its structure to support the co-execution. Besides that, as the great benefit of co-execution is providing visual feedback during the execution signalizing which entity is being assessed in each model at a given time, this process is usually slow and requires an evaluation being conducted manually to reveal its benefits. Another benefit of our approach, when compared with others, is that we defined an open and flexible architecture where different notations and tools for designing task models could fit in the future. For that, it is enough to implement a new core interface for describing the way such notations and tools deal with tasks and scenarios, and how they can be identified in their source files.

The approach has also been extended and adapted to assess other early artifacts such as GUI prototypes and late artifacts such as final GUIs [39], [34], [37], [36], [40]. As an integrated approach, the same set of BDD stories is assigned to automatically assess task models, GUI prototypes in different levels of abstraction, and final GUIs, ensuring a consistent verification, validation and testing (VV&T) approach for interactive systems with high availability of tests and instantaneous feedback about the consistency of artifacts regarding the user requirements. Strategies for running automated tests over software artifacts indeed define a step forward within the process of software verification. Such a process, that is usually conducted manually by just inspecting or reviewing the artifacts in an attempt to identify inconsistencies and modeling errors, can benefit from an automated approach giving high available instantaneous feedback about the consistency of artifacts with the user requirements all along the iterations.

Our future works include evolving this approach to manipulate and check the consistency of task models directly in their source files, besides evaluating the impact of maintaining and successively evolving the task models throughout a real software development process. Another improvement in the task model assessment that is in the pipeline consists of implementing a better technique to assess "displaying" tasks, i.e. tasks that involve the system displaying a message to the user, or the user checking that such a message has been displayed. In the current implementation, we search for a task named "Display <message>", but indeed it is not a common practice to describe system messages literally in a task name, so this kind of tasks are never matched with the equivalent steps in the BDD stories. Our first strategy in mind is to look for a generic task named "Display message", for example, and then check the actual message in the object value associated to this task. While this strategy solves the matching of equivalent tasks, if the designer skips informing the actual message when extracting a scenario from the task model, such a task would still be unidentifiable. Anyway, so far it seems to be the best strategy to address such a problem.

We are also refining our set of tools to better support the creation, visualization and execution of the tests. An important improvement as future works concerns the presentation of the task model assessment results. Despite being useful to locate exactly where the corresponding tasks have been found, a presentation based on a detailed list of matching tasks, positions and values tends to be hard to read with the growing number of scenarios. An interactive interface allowing, for example, to select and visualize only the points of inconsistencies in the models might probably help designers to evaluate and better analyzing the results.

REFERENCES

[1] Jim Barnett and et al. 2019. State Chart XML (SCXML): State Machine Notation for Control Abstraction, W3C. http://www.w3.org/TR/scxml/

[2] Judy Bowen and Steve Reeves. 2011. UI-driven test-first development of interactive systems. In *Proceedings of the 3rd ACM SIGCHI symposium on Engineering interactive computing systems*. ACM, 165–174.

[3] José Creissac Campos, Camille Fayollas, Marcelo Gonçalves, Célia Martinie, David Navarre, Philippe Palanque, and Miguel Pinto. 2017. A more intelligent test case generation approach through task models manipulation. *Proceedings of the ACM on Human-Computer Interaction* 1, EICS (2017), 9.

[4] José C Campos, Camille Fayollas, Célia Martinie, David Navarre, Philippe Palanque, and Miguel Pinto. 2016. Systematic automation of scenario-based testing of user interfaces. In *Proceedings of the 8th ACM SIGCHI Symposium on Engineering Interactive Computing Systems*. ACM, 138–148.

[5] D. Chelimsky and D. Astels. 2010. *The RSpec Book: Behaviour-driven Development with RSpec, Cucumber, and Friends*. Pragmatic Bookshelf.

[6] Elliot J. Chikofsky and James H Cross. 1990. Reverse engineering and design recovery: A taxonomy. *IEEE software* 7, 1 (1990), 13–17.

[7] Mike Cohn. 2004. *User Stories Applied: For Agile Software Development*. Addison-Wesley Professional.

[8] Rogerio Atem de Carvalho, Fernando Luiz de Carvalho e Silva, and Rodrigo Soares Manhaes. 2010. Mapping Business Process Modeling constructs to Behavior Driven Development Ubiquitous Language. arXiv:1006.4892 [cs.SE]

[9] Rogerio Atem de Carvalho, Rodrigo Soares Manhães, and Fernando Luis de Carvalho e Silva. 2010. Filling the Gap between Business Process Modeling and Behavior Driven Development. arXiv:1005.4975 [cs.SE]

[10] Clarisse Sieckenius De Souza. 2005. Semiotic engineering: bringing designers and users together at interaction time. *Interacting with Computers* 17, 3 (2005), 317–341.

[11] Anurag Dwarakanath and Shubhashis Sengupta. 2012. Litmus: Generation of Test Cases from Functional Requirements in Natural Language. In *Proceedings of the 17th International Conference on Applications of Natural Language Processing and Information Systems* (Groningen, The Netherlands) *(NLDB'12)*. Springer-Verlag, Berlin, Heidelberg, 58–69.

[12] Christof Ebert. 2011. *Global software and IT: a guide to distributed development, projects, and outsourcing.* John Wiley & Sons.

[13] Abigail Egbreghts. 2017. A Literature Review of Behavior Driven Development using Grounded Theory. In *27th Twente Student Conference on IT*.

[14] Racim Fahssi, Célia Martinie, and Philippe Palanque. 2015. Enhanced task modelling for systematic identification and explicit representation of human errors. In *Human-Computer Interaction*. Springer, 192–212.

[15] Gherkin. 2020. Gherkin Reference. https://cucumber.io/docs/gherkin/reference/

[16] Iyad Khaddam, Nesrine Mezhoudi, and Jean Vanderdonckt. 2015. Towards task-based linguistic modeling for designing GUIs. In *27ème conférence francophone sur l'Interaction Homme-Machine*. ACM, a17.

[17] G. Kotonya and I. Sommerville. 1996. Requirements engineering with viewpoints. *Software Engineering Journal* 11, 1 (Jan 1996), 5–18. https://doi.org/10.1049/sej.1996.0002

[18] Julio Cesar Sampaio do Prado Leite and Peter A. Freeman. 1991. Requirements Validation Through Viewpoint Resolution. *IEEE Trans. Softw. Eng.* 17, 12 (Dec. 1991), 1253–1269. https://doi.org/10.1109/32.106986

[19] J. C. S. P. Leite and A. P. Oliveira. 1995. A client oriented requirements baseline. In *Proceedings of 1995 IEEE International Symposium on Requirements Engineering (RE'95)*. 108–115. https://doi.org/10.1109/ISRE.1995.512551

[20] John Horn Lopes. 2012. *Evaluation of Behavior-Driven Development.* Master's thesis. Delft University of Technology.

[21] G. Lucassen, F. Dalpiaz, J. M. E. M. v. d. Werf, S. Brinkkemper, and D. Zowghi. 2017. Behavior-Driven Requirements Traceability via Automated Acceptance Tests. In *2017 IEEE 25th International Requirements Engineering Conference Workshops (REW)*. 431–434. https://doi.org/10.1109/REW.2017.84

[22] Kris Luyten, Tim Clerckx, Karin Coninx, and Jean Vanderdonckt. 2003. Derivation of a Dialog Model from a Task Model by Activity Chain Extraction. In *Interactive Systems. Design, Specification, and Verification*, Joaquim A. Jorge, Nuno Jardim Nunes, and João Falcão e Cunha (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 203–217.

[23] D. Lübke and T. van Lessen. 2016. Modeling Test Cases in BPMN for Behavior-Driven Development. *IEEE Software* 33, 5 (Sep. 2016), 15–21. https://doi.org/10.1109/MS.2016.117

[24] Célia Martinie, David Navarre, Philippe Palanque, and Camille Fayollas. 2015. A generic tool-supported framework for coupling task models and interactive applications. In *Proceedings of the 7th ACM SIGCHI Symposium on Engineering Interactive Computing Systems*. ACM, 244–253.

[25] Célia Martinie, Philippe Palanque, and Marco Winckler. 2011. Structuring and Composition Mechanisms to Address Scalability Issues in Task Models. In *IFIP Conference on Human-Computer Interaction*. Springer, 589–609.

[26] Grigori Igorovych Melnik. 2007. *Empirical Analyses of Executable Acceptance Test Driven Development.* Ph.D. Dissertation. University of Calgary. AAINR33806.

[27] Dan North. 2019. What's in a Story? http://dannorth.net/whats-in-a-story/

[28] B. Nuseibeh, J. Kramer, and A. Finkelstein. 1994. A framework for expressing the relationships between multiple views in requirements specification. *IEEE Transactions on Software Engineering* 20, 10 (Oct 1994), 760–773. https://doi.org/10.1109/32.328995

[29] Fabio Paternò. 2004. ConcurTaskTrees: An Engineered Notation for Task Models. *The handbook of task analysis for human-computer interaction* (2004), 483–503.

[30] Fabio Paternò and et al. 2017. W3C, MBUI - Task Models. http://www.w3.org/TR/task-models/

[31] Z. Pozgaj. 2000. Requirement analysis using VORD. In *ITI 2000. Proceedings of the 22nd International Conference on Information Technology Interfaces (Cat. No.00EX411)*. 105–110.

[32] Jaroslav Pullmann. 2019. MBUI - Glossary - W3C. https://www.w3.org/TR/mbui-glossary/

[33] Thiago Rocha Silva, Marco Winckler, and Cédric Bach. 2019. Evaluating the usage of predefined interactive behaviors for writing user stories: an empirical study with potential product owners. *Cognition, Technology & Work* (16 May 2019). https://doi.org/10.1007/s10111-019-00566-3

[34] T. R. Silva. 2016. Definition of a Behavior-Driven Model for Requirements Specification and Testing of Interactive Systems. In *2016 IEEE 24th International Requirements Engineering Conference (RE)*. 444–449. https://doi.org/10.1109/RE.2016.12

[35] T. R. Silva, J. Hak, and M. Winckler. 2017. A Behavior-Based Ontology for Supporting Automated Assessment of Interactive Systems. In *2017 IEEE 11th International Conference on Semantic Computing (ICSC)*. 250–257. https://doi.org/10.1109/ICSC.2017.73

[36] T. R. Silva, J-L. Hak, and M. Winckler. 2016. An Approach for Multi-Artifact Testing Through an Ontological Perspective for Behavior-Driven Development. *Complex Systems Informatics and Modeling Quarterly* 7 (June/July 2016), 81–107. https://doi.org/10.7250/csimq.2016-7.05

[37] Thiago Rocha Silva, Jean-Luc Hak, and Marco Winckler. 2016. Testing Prototypes and Final User Interfaces Through an Ontological Perspective for Behavior-Driven Development. In *Human-Centered and Error-Resilient Systems Development*, Cristian Bogdan, Jan Gulliksen, Stefan Sauer, Peter Forbrig, Marco Winckler, Chris Johnson, Philippe Palanque, Regina Bernhaupt, and Filip Kis (Eds.). Springer International Publishing, Cham, 86–107. https://doi.org/10.1007/978-3-319-44902-9_7

[38] Thiago Rocha Silva, Jean-Luc Hak, and Marco Winckler. 2017. A Formal Ontology for Describing Interactive Behaviors and Supporting Automated Testing on User Interfaces. *International Journal of Semantic Computing* 11, 04 (2017), 513–539. https://doi.org/10.1142/S1793351X17400219

[39] Thiago Rocha Silva and Marco Winckler. 2016. Towards Automated Requirements Checking Throughout Development Processes of Interactive Systems. In *Joint Proceedings of the 22nd International Conference on Requirements Engineering: Foundation for Software Quality (REFSQ 2016)*, Vol. 1564. CEUR-WS.org.

[40] Thiago Rocha Silva and Marco Winckler. 2017. A Scenario-Based Approach for Checking Consistency in User Interface Design Artifacts. In *Proceedings of the XVI Brazilian Symposium on Human Factors in Computing Systems* (Joinville, Brazil) *(IHC 2017)*. Association for Computing Machinery, New York, NY, USA, Article 3, 10 pages. https://doi.org/10.1145/3160504.3160506

[41] John Ferguson Smart. 2014. *BDD in Action: Behavior-driven development for the whole software lifecycle.* Manning Publications.

[42] H. M. Sneed. 2007. Testing against Natural Language Requirements. In *Seventh International Conference on Quality Software (QSIC 2007)*. 380–387. https://doi.org/10.1109/QSIC.2007.4385524

[43] C. Solis and X. Wang. 2011. A Study of the Characteristics of Behaviour Driven Development. In *2011 37th EUROMICRO Conference on Software Engineering and Advanced Applications*. 383–387. https://doi.org/10.1109/SEAA.2011.76

[44] Ioannis Stamelos and Alexis Tsoukias. 2003. Software evaluation problem situations. *European Journal of Operational Research* 145, 2 (2003), 273–286.

[45] Jeff Tian. 2005. *Software quality engineering: testing, quality assurance, and quantifiable improvement.* John Wiley & Sons, Chapter Software Inspection, 237—-250.

[46] Pedro Valente, Thiago Silva, Marco Winckler, and Nuno Nunes. 2016. Bridging Enterprise and Software Engineering Through an User-Centered Design Perspective. In *Web Information Systems Engineering – WISE 2016*, Wojciech Cellary, Mohamed F. Mokbel, Jianmin Wang, Hua Wang, Rui Zhou, and Yanchun Zhang (Eds.). Springer International Publishing, Cham, 349–357. https://doi.org/10.1007/978-3-319-48743-4_28

[47] Pedro Valente, Thiago Silva, Marco Winckler, and Nuno Nunes. 2017. The Goals Approach: Agile Enterprise Driven Software Development. In *Complexity in Information Systems Development*, Jerzy Goluchowski, Malgorzata Pankowska, Henry Linger, Chris Barry, Michael Lang, and Christoph Schneider (Eds.). Springer International Publishing, Cham, 201–219. https://doi.org/10.1007/978-3-319-52593-8_13

[48] Pedro Valente, Thiago Rocha Silva, Marco Winckler, and Nuno Jardim Nunes. 2016. The Goals Approach: Enterprise Model-Driven Agile Human-Centered Software Engineering. In *Human-Centered and Error-Resilient Systems Development*, Cristian Bogdan, Jan Gulliksen, Stefan Sauer, Peter Forbrig, Marco Winckler, Chris Johnson, Philippe Palanque, Regina Bernhaupt, and Filip Kis (Eds.). Springer International Publishing, Cham, 261–280. https://doi.org/10.1007/978-3-319-44902-9_17

[49] Rudolf Van Megen and Dirk B Meyerhoff. 1995. Costs and benefits of early defect detection: experiences from developing client server and host applications. *Software Quality Journal* 4, 4 (1995), 247–256.

[50] Marco Winckler and Philippe Palanque. 2003. StateWebCharts: A Formal Description Technique Dedicated to Navigation Modelling of Web Applications. In *Interactive Systems. Design, Specification, and Verification.* Springer Berlin Heidelberg, 61–76.

[51] Marco Winckler and Philippe Palanque. 2012. Models as Representations for Supporting the Development of e-Procedures. In *Usability in Government Systems.* Elsevier, 301–315.

[52] Andreas Wolff, Peter Forbrig, Anke Dittmar, and Daniel Reichart. 2005. Linking GUI elements to tasks: supporting an evolutionary design process. In *Proceedings of the 4th international workshop on Task models and diagrams.* ACM, 27–34.

Received October 2019; Revised November 2019; Accepted December 2019