Behaviour-Driven Development for Computer-Interpretable Clinical Guidelines

Reinhard Hatko[1], Stefan Mersmann[2], and Frank Puppe[1]

[1] Institute of Computer Science, University of Würzburg, Germany
[2] Dräger Medical GmbH, Lübeck, Germany

Abstract. In this paper, we propose an approach for specification and analysis of Computer-Interpretable Clinical Guidelines (CIGs) that was inspired by Behaviour-Driven Development. The expected behaviour of a CIG is specified by requirements in natural language. On the one hand, those requirements are used as design input for guideline development. On the other hand, they can be checked against time-oriented data recorded during testing sessions of the implemented CIG.

1 Introduction

Continuously improving quality and efficiency of health care delivery are major goals in medicine, especially in critical care. Approaches from process engineering that identify, organize and standardize therapeutic workflows have been established to meet these goals. Evidence-based workflows in medicine are called clinical guidelines (CGs). CGs are standardized therapeutic plans that reflect best practices for treating patients within a certain medical scope. The impact of CGs on patient outcome had been investigated and published through several clinical studies, e.g., [5]. A logical next step was the integration of standardized health care processes into medical technology by enabling CGs to be executed by medical devices as Computer-Interpretable Clinical Guidelines (CIGs).

Recently, we demonstrated the applicability of the Semantic Wiki KnowWE for the development of a CIG for automated mechanical ventilation [4]. The visual CIG language DiaFlux [3] was used to implement the guideline logic. However, the described development process lacked a specification of CIG behaviour at an abstract level on which the implementation was based.

The rest of the paper describes an approach to fill this gap: The next section introduces Behaviour-Driven Development (BDD) in Software Engineering. Section 3 shows its applicability in the area of CIGs. A case study in progress is presented in Section 4. The paper concludes with a summary in Section 5.

2 Behaviour-Driven Development

Behaviour-Driven Development (BDD) is an agile Software Engineering (SE) methodology, that conforms to the *Test-First* principle: Test cases are developed previous to the software system itself. In BDD, executable test cases are directly derived from requirements stated in natural language, by employing a pattern matching approach. This enables stakeholders to actively participate in the definition of requirements, which are a fundamental design input for the system under development.

In contrast to other Test-First approaches (like Test-Driven Development), BDD focuses on the creation of acceptance tests rather than unit tests. The former ones assure the acceptance of the system by its users, as the system's ability to fulfill their needs with respect to a *business value* is tested. The latter ones perform testing on a lower system level by checking individual program constructs. This has the potential to reduce the number of defects at an early stage of the software development process.

```
Scenario: trader is not alerted below threshold
    Given a stock of symbol STK1 at a threshold of 10.0
    When the stock is traded at 5.0
    Then the alert status should be OFF

Scenario: trader is alerted above threshold
    Given a stock of symbol STK1 at a threshold of 10.0
    When the stock is traded at 11.0
    Then the alert status should be ON
```

Fig. 1: Two scenarios from the stock market domain, expressed in Gherkin.

BDD frameworks exist for several programming languages, e.g. JBehave[3] (Java) or RSpec [1] (Ruby). As a commonality, those frameworks offer some kind of Domain-Specific Language (DSL) for defining scenarios, and a mechanism to derive executable test code by some sort of pattern matching, usually regular expressions. One such DSL which is employed in multiple frameworks is called the *Gherkin* language [1]. Figure 1 shows an exemplary usage of Gherkin. The scenarios consist of the following three groups:

- The keyword `GIVEN` sets up the test environment by creating the specific context which is necessary to execute the test.
- Second, the functionality under test is executed (keyword `WHEN`).
- Lastly, the outcome of the previous step is compared against the expected one (keyword `THEN`).

Each group can contain several steps that are joined together using the keyword `AND`. Each step in turn, is backed by a step template (also called *support code*), that converts the text into executable program code using regular expressions, e.g. to extract method parameters. Figure 2 shows the step templates needed to execute both scenarios shown in Figure 1.

As each step is individually backed by support code, steps can arbitrarily be combined to create new acceptance tests. Thanks to the comprehensibility of Gherkin and the use of natural language, scenarios can easily be created also

[3] http://jbehave.org

```
public class TraderSteps {
    private Stock stock;

    @Given("a stock of symbol $sym at a threshold of $thr")
    public void aStock(String sym, double thr) {
        stock = new Stock(sym, thr); }

    @When("the stock is traded at $price")
    public void theStockIsTradedAt(double price) {
        stock.tradeAt(price); }

    @Then("the alert status should be $stat")
    public void theAlertStatusShouldBe(String stat) {
        ensureThat(stock.getStatus().name(), equalTo(stat)); }
}
```

Fig. 2: The according support code for the scenarios in Figure 1, expressed in Java and JBehave.

by non-programmers, e.g. the future users of the software themselves. That way, a *ubiquitous language* [2] is created, that improves communication between all participants involved in the software development process, e.g., product owner, software engineers, testers, and users.

3 Specification of Clinical Guidelines

In the remainder of this section, we describe an approach for the specification of automated CIGs following the BDD methodology. The CIG behaviour is specified as a set of scenarios expressed in the Gherkin language:

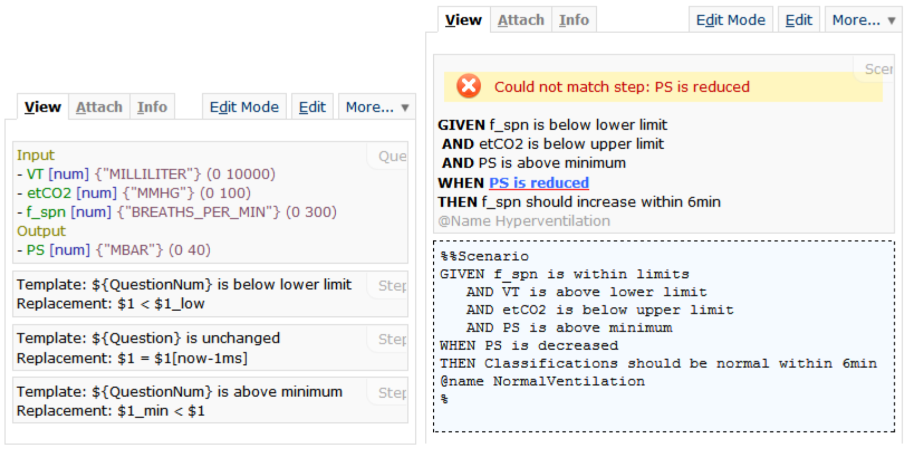
- The `GIVEN` group of a scenario describes the precondition in terms of the patient's current state, and possibly also its past progression.
- The expected therapeutic action, i.e. the output of the CIG for the given patient state, is expressed in the `WHEN` part of the scenario.
- The `THEN` group contains assumptions about the future (improved) patient state, based on the therapeutic action. As the effects will take some time until they are measurable, usually a temporal annotation is included.

There is a main difference between the SE acceptance tests as described in the previous section and the CIG scenarios. While the former ones are *actively* preparing the test context and executing the function under test, the latter ones can only *"passively"* be evaluated on data a patient delivered. The CIG scenarios are interpretable descriptions of patterns that are supposed to occur in patient data, given that the CIG has correctly been implemented based on an error-free specification.

After implementing the CIG based on the defined scenarios, test cases can be recorded, e.g. using a patient simulator [4]. The data consists of time-stamped recordings of the patient state and the according therapy over the course of the session. Therapeutic actions are changes of the medical device's settings applied automatically according to the underlying CIG. By checking the scenarios against a test case, two types of errors can be discovered: First, the designated therapeutic action (`WHEN`-group) may not occur due to a bug in the implementation ("error of omission"), although the current patient state fulfills the precondition of the scenario (`GIVEN`-group). Second, if the precondition as well as the action of a scenario are met, but not the expected change in the patient's state, the assumptions about the state may not be correct in the first place. While the first error is most likely rooted in the implementation of the CIG, the second kind of error may arise from an improper specification itself.

4 Case Study

We have implemented an extension for the Semantic Wiki KnowWE that supports the described approach. Each step template consists of a regular expression containing capturing groups enriched by a type system, and a formal condition with the according placeholders. When using step templates, the placeholders are filled with terms from a predefined ontology, that contains the data definitions of the CIG (cf. Figure 3, left side). The ontology is created as the foundation for the fully formalized CIG [4]. Scenarios are expressed by arbitrarily combining steps. If a step can not be matched against a template, an error is reported. This leads to the definition of new templates, incrementally increasing the range of expressible scenarios as needed (cf. Figure 3, right side).



```
Input
- VT [num] {"MILLILITER"} (0 10000)
- etCO2 [num] {"MMHG"} (0 100)
- f_spn [num] {"BREATHS_PER_MIN"} (0 300)
Output
- PS [num] {"MBAR"} (0 40)
```

```
Template: ${QuestionNum} is below lower limit
Replacement: $1 < $1_low
```

```
Template: ${Question} is unchanged
Replacement: $1 = $1[now-1ms]
```

```
Template: ${QuestionNum} is above minimum
Replacement: $1_min < $1
```

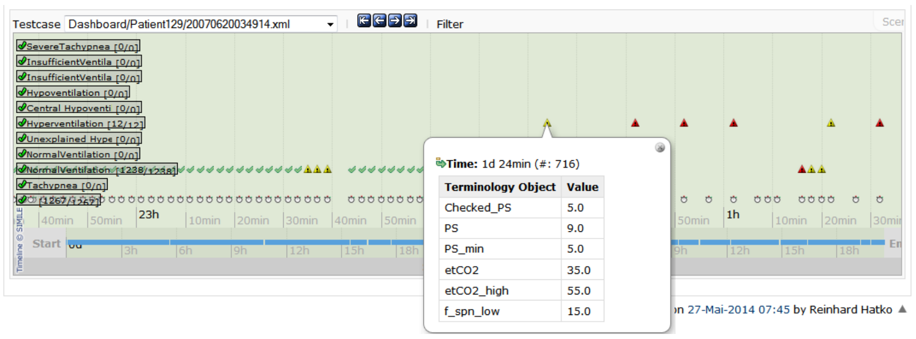
Could not match step: PS is reduced

GIVEN f_spn is below lower limit
AND etCO2 is below upper limit
AND PS is above minimum
WHEN PS is reduced
THEN f_spn should increase within 6min
@Name Hyperventilation

```
%%Scenario
GIVEN f_spn is within limits
   AND VT is above lower limit
   AND etCO2 is below upper limit
   AND PS is above minimum
WHEN PS is decreased
THEN Classifications should be normal within 6min
@name NormalVentilation
%
```

Fig. 3: Two KnowWE wiki pages: The left page contains parts of the data ontology and step templates. The right one contains two scenarios in display and in editing mode. Steps that cannot be matched result in an error.

The results of the scenario analysis with respect to a test case are visualized on an interactive timeline, that can be panned and zoomed, cf. Figure 4. Each band corresponds to exactly one scenario. At each instant of time, an icon represents the result, if the scenario's precondition is fulfilled: omitted therapeutic action (red warning sign), unfulfilled postcondition (yellow warning sign), and fully satisfied scenario (green checkmark), respectively. The timeline is integrated into KnowWE's testing framework. Replaying the test case to any instant in time is possible to introspect CIG execution for debugging purposes.



Time: 1d 24min (#: 716)

Terminology Object	Value
Checked_PS	5.0
PS	9.0
PS_min	5.0
etCO2	35.0
etCO2_high	55.0
f_spn_low	15.0

Fig. 4: Analysis results of the defined scenarios with respect to a temporal test case are depicted on an interactive timeline. For debugging purposes, the test case can be replayed until the currently selected instant of time by clicking the green arrow inside the tooltip.

For the evaluation of the presented approach, we are currently working on a case study with a real world CIG and patient data from a previous study conducted at the Department of Anesthesiology and Intensive Care Medicine, University Medical Center Schleswig-Holstein, Campus Kiel [8]. During this study, 150 patients have been automatically weaned from mechanical ventilation by the automatic weaning system SmartCare/PS® [6]. SmartCare/PS is a knowledge-based software option for Dräger's mechanical ventilators Evita XL and Evita Infinity V500. SmartCare/PS stabilizes the patient's spontaneous breathing in a respiratory comfort zone, while gradually reducing the inspiratory support of the ventilator until the patient is ready for extubation. The system's underlying CG is further depicted in [7].

This study does not investigate the usage of the derived specification as a design input. It focuses on the applicability of the approach in terms of usability and expressiveness, and its usage for analysing the test cases regarding the scenarios. The CIG behaviour has been specified using natural language scenarios as described herein. By exploiting strict naming conventions regarding the labeling of related data, e.g. measurements and their corresponding upper and lower limits, only a rather limited set of about 15 step templates needed to be defined. This allows for fast requirements elicitation and also reduces maintenance efforts.

5 Conclusion

In this paper, we have described an approach inspired by Behaviour-Driven Development for specification and analysis of Computer-Interpretable Clinical Guidelines. Requirements stated by medical experts in natural language are used as design input for the development of CIGs and their analysis using test cases. We demonstrated the applicability of Behaviour-Driven Development for CIGs using an extension for the Semantic Wiki KnowWE. Currently, we are conducting a case study focusing on the analysis aspect. A real world CIG and test cases from a real patient study are used for evaluation. So far, the approach has shown its applicability in terms of usability and expressiveness. In the near future, the results of the scenarios will be analysed by medical experts.

References

1. Chelimsky, D., Astels, D., Dennis, Z., Hellesoy, A., Helmkamp, B., North, D.: The RSpec Book: Behaviour Driven Development with RSpec, Cucumber, and Friends. The Pragmatic Programmers, United States (2011)
2. Evans, E.: Domain-Driven Design: Tackling Complexity in the Heart of Software. Addison-Wesley Longman, Amsterdam, 1 edn. (2003)
3. Hatko, R., Baumeister, J., Belli, V., Puppe, F.: DiaFlux: A graphical language for computer-interpretable guidelines. In: Riaño, D., ten Teije, A., Miksch, S. (eds.) Knowledge Representation for Health-Care, Lecture Notes in Computer Science, vol. 6924, pp. 94–107. Springer, Berlin / Heidelberg (2012)
4. Hatko, R., Schädler, D., Mersmann, S., Baumeister, J., Weiler, N., Puppe, F.: Implementing an Automated Ventilation Guideline Using the Semantic Wiki KnowWE. In: ten Teije, A., Völker, J., Handschuh, S., Stuckenschmidt, H., d'Aquin, M., Nikolov, A., Aussenac-Gilles, N., Hernandez, N. (eds.) EKAW. Lecture Notes in Computer Science, vol. 7603, pp. 363–372. Springer (2012)
5. Lellouche, F., Mancebo, J., Jolliet, P., Roeseler, J., Schortgen, F., Dojat, M., Cabello, B., Bouadma, L., Rodriguez, P., Maggiore, S., Reynaert, M., Mersmann, S., Brochard, L.: A multicenter randomized trial of computer-driven protocolized weaning from mechanical ventilation. American journal of respiratory and critical care medicine 174(8), 894–900 (Oct 2006)
6. Mersmann, S., Dojat, M.: SmartCare/PS-Automated Clinical Guidelines in Critical Care. In: de Mántaras, R.L., Saitta, L. (eds.) Proceedings of the 16th Eureopean Conference on AI, ECAI'2004. pp. 745–749. IOS Press 2004, Valencia, Spain (2004)
7. Neumann, A., Schmidt, H.: SmartCare®/PS The automated weaning protocol (2010), http://www.draeger.net/media/10/01/35/10013505/smartcare_auto_weaning_protocol_booklet.pdf
8. Schädler, D., Engel, C., Elke, G., Pulletz, S., Haake, N., Frerichs, I., Zick, G., Scholz, J., Weiler, N.: Automatic control of pressure support for ventilator weaning in surgical intensive care patients. American journal of respiratory and critical care medicine 185(6), 637–44 (Mar 2012)