From Behaviour-Driven Development Scenarios to Formally Verifiable Behavioural Models via Dynamic Condition Response Graphs (extended abstract)

Xinyuan Tu[1], Thomas Hildebandt[1], and Thiago Rocha Silva[2]

[1] University of Copenhagen, Universitetsparken 5, 2100 Copenhagen, Denmark
[2] University of Southern Denmark, Campusvej 55, 5230 Odense, Denmark
{xinyuan.tu,hilde}@di.ku.dk
thiago@mmmi.sdu.dk

Model-Driven Software Engineering (MDSE) based on formal, verifiable models has been driven by the promises of being able to generate reliable and provably correct software systems from models. However, some of the key challenges for the success of MDSE have been the need for learning formal modelling notations and also to reduce the gap among the formal modelling notations, traditional requirement specifications, the business understanding of the requirements and eventually deploying the software systems to end-users. Behaviour-Driven Development (BDD) [4] has been introduced as a means to address these challenges, bridging between business and technical experts by providing a natural-language domain-specific language for describing the behaviour of software systems. Some authors have studied the (semi-) automated generation of formal models from Behaviour-Driven Development DSLs [15,9].

In the position paper [13] the last two authors of the present extended abstract introduced a research agenda for reliable and explainable model-driven software engineering pursued in the recently initiated PhD project of the first author. As shown in Fig. 1 below (reproduced from [13]), the key idea of the proposed approach is that an LLM, configured by software engineers, could be used to translate natural language requirements to an intermediate DSL, which could then be validated by end-users (domain-experts) and automatically translated to process models. The position paper focuses on so-called process-aware information systems (PAIS) [11], i.e., information systems supporting processes based on explicit process models that are either compiled to code or interpreted when the system is used. Dynamic Condition Response (DCR) graphs [6] is chosen as the target process modelling language. DCR graphs are already successfully used in industry [5] and are supported by end-user tools for simulation. Moreover, the DCR graph language has a formal semantics that opens for formal verification of the models. Two prior PhD projects [3,2,10] provide mappings of the core DCR modelling language to Büchi-automata and 1-safe Petri Nets, and of the timed extension of DCR Graphs to Büchi-automata with time-ticks or Timed-Arc Petri Nets [8].

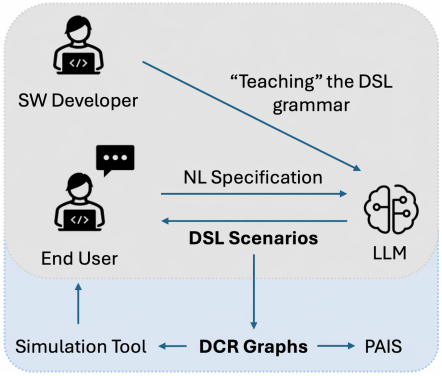

Fig. 1. The proposed approach.

The position paper illustrated that the example of a vending machine in the DSL proposed in [12,9] could be represented as a timed DCR Graph [7]. We recall the excerpt of the DSL and the resulting (hand-crafted) DCR Graph in the appendix in Fig. 2. Since the Behaviour-Driven Development of [9] is based on states and scenarios, describing how actions transit between states and there are very few existing works combining process modelling and Behaviour-Driven Development, this interests us to find whether the Behaviour-Driven Development could be applied to describe non-state based process mining domain (e.g. DCR graph).

In our ongoing work we investigated the development of an automated generation of DCR Graphs from the DSL-language. We explore two approaches: The first approach is the generation of automata-like DCR graphs as suggested in [13]. These graphs can be mapped to Timed-Arc Petri Nets and verified in the TAPAAL tool or mapped to timed automata [14]. The second approach is to work on the design of a new DSL that is closer to the particular, declarative primitives of DCR Graphs. Our hypothesis is that the second approach will be more suitable in situations, where the natural language requirement to be translated is not formulated in terms of states or modes of a machine. The latter approach involves introducing declarative specifications of the model states inspired by how states are described in DCR Graphs, i.e. specifying the properties of a state in terms of what actions are available and what actions are pending. Finally, we have also initiated work on extending the DSL to capture both data and *object creation* and mapping to object-centric DCR Graphs [1,16] to be able to represent object oriented systems. Such models are no longer guaranteed to be finite state, but we are planning to investigate different approaches to verification, e.g. assuming bounded data types and generation of objects. Verification can be done either developing custom DCR verification tools or by mapping DCR Graphs to existing other formal models such as timed-arc Petri Nets or timed-automata [14]. All approaches will be combined with conversational LLMs using human-centric way to facilitate mapping of natural language requirements to Behaviour-Driven Development specifications by end-users, which can then be mapped to DCR graphs and simulated using the DCR Graph design tools [3] and formally verified.

The talk at the workshop will present ongoing work based on examples and discuss related work and possible paths for future work.

[3] A free academic user license can be created at dcrsolutions.net

References

1. Christfort, A.K., Rivkin, A., Fahland, D., Hildebrandt, T.T., Slaats, T.: Discovery of object-centric declarative models. In: 2024 6th International Conference on Process Mining (ICPM). pp. 121–128 (2024). https://doi.org/10.1109/ICPM63005.2024.10680680
2. Cosma, V.P.: Declarative process models as explainable and verifiable Artificial Intelligence. Ph.D. thesis, Copennhagen University (June 2024), https://di.ku.dk/english/research/phd/phd-theses/2024/Vlad_paul_phd.pdf
3. Cosma, V.P., Hildebrandt, T.T., Slaats, T.: Transforming dynamic condition response graphs to safe petri nets. In: Gomes, L., Lorenz, R. (eds.) Application and Theory of Petri Nets and Concurrency. pp. 417–439. Springer Nature (2023)
4. Farooq, M.S., Omer, U., Ramzan, A., Rasheed, M.A., Atal, Z.: Behavior driven development: A systematic literature review. IEEE access **11**, 88008–88024 (2023)
5. Hildebrandt, T.T., Andaloussi, A.A., Christensen, L.R., Debois, S., Healy, N.P., López, H.A., Marquard, M., Møller, N.L.H., Petersen, A.C.M., Slaats, T., Weber, B.: Ecoknow: Engineering effective, co-created and compliant adaptive case management systems for knowledge workers. In: Proceedings of the International Conference on Software and System Processes. p. 155–164. ICSSP '20, Association for Computing Machinery, New York, NY, USA (2020). https://doi.org/10.1145/3379177.3388908, https://doi.org/10.1145/3379177.3388908
6. Hildebrandt, T.T., Mukkamala, R.R.: Declarative event-based workflow as distributed dynamic condition response graphs. In: Proceedings of the Third Workshop on Programming Language Approaches to Concurrency and Communication-cEntric Software, PLACES 2010. EPTCS, vol. 69, pp. 59–73 (2010)
7. Hildebrandt, T.T., Mukkamala, R.R., Slaats, T., Zanitti, F.: Contracts for cross-organizational workflows as timed dynamic condition response graphs. J. Log. Algebr. Program. **82**(5-7), 164–185 (2013)
8. Jacobsen, L., Jacobsen, M., Møller, M.H., Srba, J.: Verification of timed-arc petri nets. In: Černá, I., Gyimóthy, T., Hromkovič, J., Jefferey, K., Králović, R., Vukolić, M., Wolf, S. (eds.) SOFSEM 2011: Theory and Practice of Computer Science. pp. 46–72. Springer Berlin Heidelberg, Berlin, Heidelberg (2011)
9. Kang, E.Y., Silva, T.R.: Towards formal verification of behaviour-driven development scenarios using timed automata. In: 2023 30th Asia-Pacific Software Engineering Conference (APSEC). pp. 612–616. IEEE (2023)
10. Mukkamala, R.R.: A Formal Model For Declarative Workflows: Dynamic Condition Response Graphs. Ph.D. thesis, IT University of Copenhagen (June 2012)
11. Reichert, M., Weber, B.: Enabling Flexibility in Process-Aware Information Systems - Challenges, Methods, Technologies. Springer (2012)
12. Silva, T.R.: Towards a domain-specific language for behaviour-driven development. In: IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC 2023. pp. 283–286. IEEE (2023)
13. Silva, T.R., Hildebrandt, T.T.: Human-centric hybrid-AI for no-code development. In: Proceedings of the 33rd ACM International Conference on the Foundations of Software Engineering. p. 1520–1521. FSE Companion '25, Association for Computing Machinery, New York, NY, USA (2025). https://doi.org/10.1145/3696630.3731669, https://doi.org/10.1145/3696630.3731669
14. Srba, J.: Comparing the expressiveness of timed automata and timed extensions of petri nets. In: Cassez, F., Jard, C. (eds.) Proc. of. FORMATS '08. LNCS, vol. 5215, pp. 15–32. Springer (2008)

15. Zameni, T., van Den Bos, P., Tretmans, J., Foederer, J., Rensink, A.: From bdd scenarios to test case generation. In: 2023 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW). pp. 36–44. IEEE (2023)
16. Zuckmantel, T., Zhou, Y., Düdder, B., Hildebrandt, T.: Daceo: Declarative asynchronous choreographies with general data-dependent event-ordering and objects. In: Coordination Models and Languages. 27th IFIP WG 6.1 International Conference, COORDINATION. Lille, France (June 2025)

Examples and informal introduction to DCR Graphs

In listing. 1.1 below is shown the example entity model of the vending machine and scenario from [9].

```
entity vending machine {
    actions: insert, select, pick, cancel, unpicked
    states: idle, dispensing mode, selection mode,
        extra dispensing mode
    properties: option, inserted amount, product
}

Scenario: Dispensing drink with extras
Given the vending machine is in selection mode
When I select the product "Coke" within 5 seconds
And I concurrently select the option "ice" on the
    vending machine
Then the vending machine is in extra dispensing mode
```

Listing 1.1. DSL entity model and example scenario of the vending machine.

Fig. 2 shows the hand-crafted DCR graph for the vending machine presented in [13] (slightly modified to include both initiator and observer roles and avoid the concept of sub processes) and it is represented graphically in the online design with its simulation tool available at DCRSolutions.net. If one registers with an university email and affiliation it is possible to get a free academic license, access the model using the link https://www.dcrgraphs.net/Tool?id=2009164 and simulate the model using the link https://sim.dcrgraphs.net?code=b4dfe8c6-abee-41ef-869b-345deaf840a4.[4]

The nodes of the DCR Graph are either *(nested) groups* of activities (represented graphically as boxes with an n in the lower right corner) or *activities* (represented as boxes with an activity label in the middle). The edges between nodes denote logical relations between activities of different types, either constraining when the activity can be executed or declaring that executing an activity has an effect on another activity. An edge to a group of activities is a compact way to represent edges to all activities (or groups) inside the group.

[4] Note that only the first link allows for copy and adapt the model, and detailed simulation where also the time can be controlled.

Activities can have one or more *roles* associated. The initiator/write roles are shown in a bar at the top of the activity, indicating which roles can execute the activity. The observer/read roles are shown in a bar at the bottom of the activity, indicating which roles can observe that the activity is executed.

The DCR Graph in Fig. 2 has 8 activities and 4 groups of activities. The activities (resp. groups) are labelled with the 5 names of actions (resp. states) shown in the entity model above. Note that there can be two or more activities having the same label, allowing the same action to have different effects when executed.

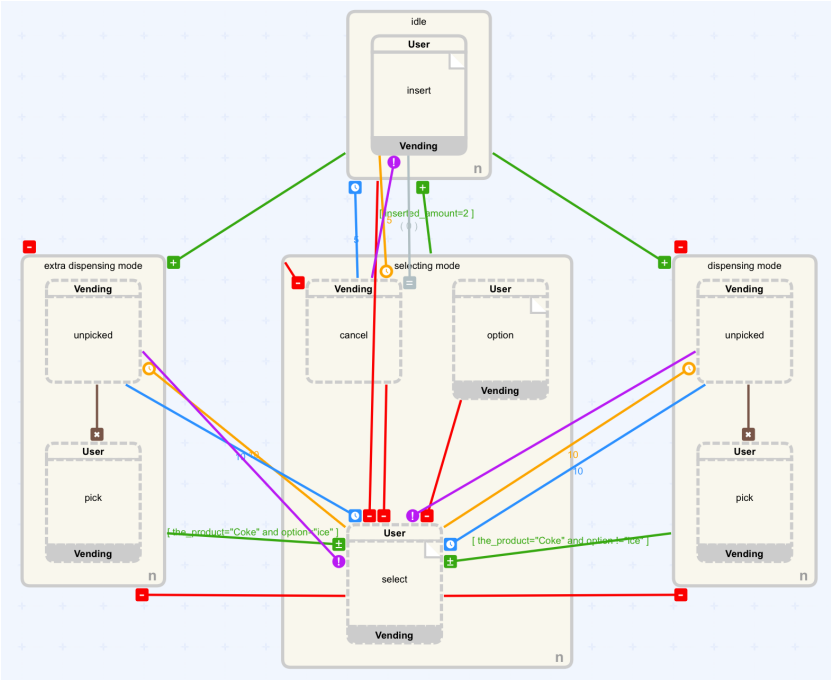

Fig. 2. Hand-crafted DCR Graph for the vending machine (slightly modified from [13]).

A DCR Graph has a formal execution semantics and a run-time state represented as a marking of the nodes in the graph. The state, referred to as the marking of the graph, consists of three elements. The first element of the state of a DCR Graph is that each activity can either be *included* or *excluded* in the current state. Included (resp. excluded) activities are shown graphically with a solid (resp. dashed) border. A necessary (but not sufficient) requirement for an activity to be enabled for execution is that it is included. In the example graph, only the activity labelled with the insert action is included. The other requirement for an activity to be enabled is that all its preconditions are either executed or excluded. Preconditions are represented with edges with a circle

at one end. There are two different preconditions in the example diagram. The orange precondition between the two unpicked activities and the select activity with the small clock inside the circle and the number 10 denote a delayed condition, which means that the unpicked activities are only enabled if they are included and at least 10 time units have passed since the select activity was last executed. In order to test for this requirement, the second element of the state of a DCR Graph is to record for each activity if it has been executed and how long ago it happened. This is shown graphically with a small checkmark in the upper left corner of the executed activities, as seen in Fig. 3, showing the graph with the updated marking after the insert activity has been executed (the time passed since the execution is not shown on the graph in the tool). The second kind of precondition is the *milestone* relation, indicated by a (purple) edge with a solid circle at one end containing an exclamation mark. The milestone relation between the insert activity and the cancel activity represents the constraint that the insert activity is not enabled for execution if the cancel activity is included and pending for execution. The leads us to the last element of a DCR Graph marking, which for each activity records whether it is *pending* for execution in the future, and possibly a deadline for when it must be executed. Graphically, pending activities are shown with an exclamation mark in the upper left corner, as shown in Fig. 3 below for the cancel activity. Activities may either be initially

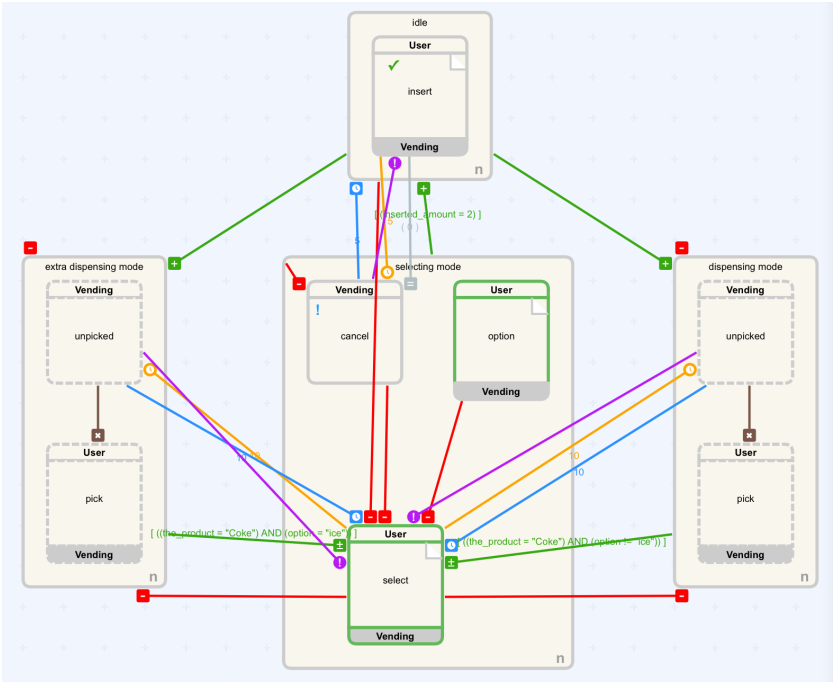

Fig. 3. DCR Graph for the vending machine after executing the insert activity.

pending for execution (none of the activities are initially pending in the vending machine graph in Fig. 2) or they may become pending for execution because of the *response* relation between activities. A response relation is a blue relation with a box at one end, containing either an exclamation mark or a clock. The response relation between the idle group and the cancel activity denotes that if any activity in the idle group is executed (e.g. insert) then the cancel activity becomes pending with the deadline of 5 time units (written on the edge, but slightly difficult to see). Similarly, the response relation from the select activity to the unpicked activity puts a 10 time unit deadline on the unpicked activities after the last execution of the select activity. Dually to the response relation, the brown edge with a star inside a box in one end denotes a *cancel* relation, which makes it possible to make an activity *not* pending. Thus, the cancel relations between the two pick activities and the corresponding unpick activity indicate that if the item is picked, then the unpick activity (vending machine automatically returns the item if not picked up) is no longer pending. Finally, the green and red edges represent include and exclude relations, respectively, which enable activities to be dynamically included or excluded. For instance, the include relation between the idle group and the selecting mode group means that when insert (in the idle group) is executed, then all the activities in the selecting mode group are included. This can be seen in Fig. 3 by the three activities now having a solid border. However, note that only the option and select activities are enabled (shown with a green border), since the *cancel* activity has a condition relation to the insert activity with a time delay of 5 time units (preventing the vending machine from canceling before 5 time units has passed since the user inserted the coin). Dually to the include relation, the exclude relation between the cancel activity and the selecting mode group denotes that all three activities in the selecting mode group get excluded if the cancel activity is executed.

The graph also uses data activities (i.e. activities that take a data value as input when executed) and data guards on some of the relations. Concretely, the insert activity takes an integer value and the select activity takes an enumeration (i.e. Coke or Pepsi). The constraint on the include relation between the idle group and the selecting mode group then specifies that include only takes effect if the right amount of 2 coins is inserted.

The formal semantics of DCR Graphs with data can be found in [16].