Forms are declarative processes!

Marquard, Morten; Debois, Søren; Slaats, Tijs; Hildebrandt, Thomas

Publication date:
2016

Document version
Early version, also known as pre-print

Citation for published version (APA):
Marquard, M., Debois, S., Slaats, T., & Hildebrandt, T. (2016). *Forms are declarative processes!*. Paper presented at Business Process Management Forum 2016, Rio de Janeiro, Brazil.

Forms are declarative processes!

Morten Marquard[1], Søren Debois[1,2], Tijs Slaats[3], and Thomas Hildebrandt[2]

[1] Exformatics A/S
{mmq,debois}@exformatics.com
[2] IT University of Copenhagen, Denmark
{debois,hilde}@itu.dk
[3] Department of Computer Science, University of Copenhagen, Denmark
slaats@di.ku.dk

Abstract. Modern form-based end-user interfaces are highly variable and adapt to the choices made by the user. Describing such adaptations programmatically is time and resource intensive and therefore more generic approaches are preferable. Form-based UIs are also often very flexible: users have a large degree of freedom in the order in which fields are filled in, and may change previous values on the fly without needing to re-do intermediate steps. In this paper, we demonstrate how such flexible behaviour can be efficiently captured via a declarative process model, where the declarative rules define for each form element whether it is visible, editable and/or mandatory. Specifically, we show (a) how to express complex, variable forms as declarative processes; (b) how to express such form processes in the DCR formalism; (c) how to model and execute such forms on the dcrgraphs.net process portal; and (d) how Exformatics is in the process of applying this approach to an ACM system delivered to one of its customers.

1 Introduction

Forms are ubiquitous in contemporary user interfaces; in adaptive case management systems perhaps even more so than in general. Although forms are well-understood both conceptually (as UI metaphors) and technically (as objects of implementation)—see, e.g., [22,11,12]—traditional methods for developing and maintaining forms retain the usual problems of software development: They are sometimes overly rigidly specified; updating or changing them typically requires expensive and time-consuming programmer intervention; yet they typically *must* change at regular intervals in response to changes in regulations and best practices of the organisation using them.

Forms seem to share certain properties with the workflows which they support, especially within the world of adaptive case management: they typically afford the user a measure of flexibility, in that they do not impose rigid rules on the order in which fields are filled in; they embody certain rules, such as "if this field is filled in, this other field must also be filled in"; and they are continuously updated (or should be!) in response to changes in regulations and best practices of their organisational context.

In this paper we will show:

- How to express complex, variable forms as declarative processes.
- How to express such form processes in the DCR formalism.
- How to model and execute such forms on the DCRGraphs.net process portal.
- How we applied this approach to the grant application system of a Danish foundation

2 Situation faced

For Exformatics A/S—who we believe are representative of industry as a whole in this instance—the implementation and especially maintenance of forms is also a practical challenge: contemporary implementation techniques invariably involve expensive bespoke programming to accommodate the whims of Exformatics' customers; and customers continuously request minor and major updates and changes to deployed forms.

Both initially implementing and subsequently updating form implementations is expensive and time-consuming. Forms are in this sense a microcosm of larger system implementation and maintenance: Customers are not necessarily precise in their initial requirements; implementation is slow and expensive; and change requests invariably follow.

We observe that both the circumstances surrounding forms as well as the implementation challenges of forms resemble the circumstances and challenges of adaptive case management. In this paper, we report on an approach to leveraging this observation by *considering forms declarative processes* and implementing them as such: one specifies not a form but a declarative workflow, and from this workflow, a form automatically appears.

The approach is motivated by practical challenges at a particular Exformatics customer; it has been implemented in the declarative workflow modelling and simulation tool, DCRGraphs.net [23,19]; and will shortly be rolled out at the customer.

In Figure 1 below is an example of a form: A dropdown box allows a choice between "Approve" or "Reject". Reject is currently selected, and a text box provides space for a short text describing the reason for the rejection. We can imagine this form being part of a travel reimbursement workflow, where a manager must either approve or reject a request for reimbursement, and is apparently expected to justify rejections.

What is perhaps not immediately apparent from Figure 1 is the dynamic, rule-based nature of forms. To support the above workflow, the form should actually have the following dynamic behaviour:

1. Initially, the form should give simply a choice between "Approve" and "Reject"—it might look like Figure 2.
2. Without a choice, the form can be "closed", but not "submitted"; that is, the manager must make a choice for the underlying process to proceed.

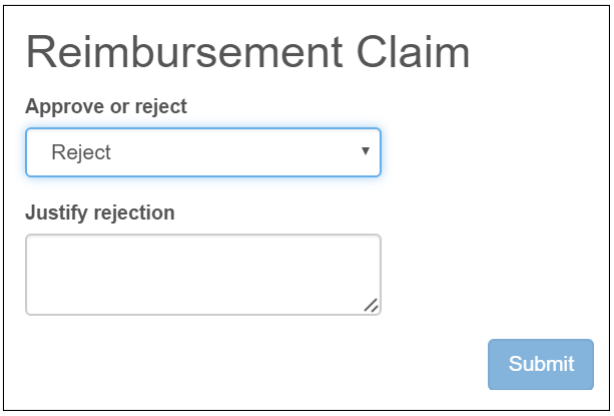

Fig. 1. Reimbursement claim approval form (rejecting).

3. If the manager chooses "Approve", the form is complete and may be submitted.
4. If the manager chooses "Reject", a new field appears—the "Justify rejection" text box of Figure 1.
5. When this field appears, it must be filled in before the form can be submitted.
6. If the manager changes his mind and reverts his choice from "Reject" to "Approve", the description field should disappear again.

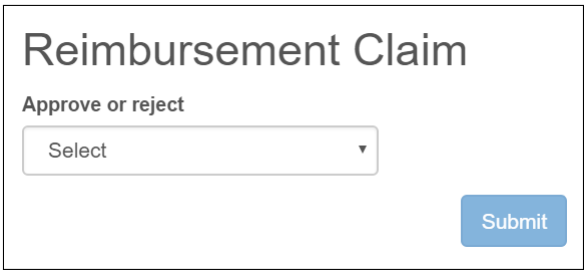

Fig. 2. Reimbursement claim approval form (initial appearance).

We make two notes about this form and its behaviour.

First, even this seemingly exceedingly simple form has fairly complex behaviour when you sit down and write it out as we did in the above list. Getting this behaviour right is not necessarily difficult, but it is time-consuming and expensive because of the required programmer intervention.

Second, the form and the rules governing it are inextricably linked with the process to which the form contributes. The little list above is littered with men-

tions of business objectives or process rules which the form happen to express, e.g., the choice of the manager in (1), or the requirement that the justification field is filled-in in (5)—the latter corresponds directly to the business rule that *A reimbursement claim can be rejected only if it the rejection is accompanied by a justification.*

We propose in this paper to specify forms using a declarative process notation. This approach has dual advantages, falling out of the above two observations:

1. If the workflow to which the form contributes is specified in a formal notation, *the form itself can be extracted from this specification.*
2. Workflows not withstanding, the mechanics of a form—fields appearing and disappearing, transitioning between being required and optional and so forth—seems *well-suited for specification in a declarative workflow notation.*

We will in subsequent sections consider how forms can be expressed as DCR graphs [13,20,24,6,8]. In the present section, we will consider what state a formalism needs to express in the general case in order to be useful for modelling forms.

Looking at the above example, it seems that the state of a form field can be described by just four attributes:

- *visibility*, i.e., is the field currently displayed in the form?
- *requiredness*, i.e., if the field is displayed, is it then mandatory to fill in, i.e., is it a requisite for submitting the form?
- *value*, i.e., if the field is displayed, has it been executed and given a value to be displayed?
- *enabledness*, i.e., if the field is displayed, is it editable?

As it happens, these attributes correspond neatly with the state of activities in the DCR graphs notation, where an activity has the attributes:

- *included* (visibility), i.e., is the activity included in the current state of the workflow?
- *pending* (requiredness), i.e., is the activity required to be executed in the future (or excluded) in order for the workflow to be considered complete?
- *executed*, i.e., has the activity been executed previously?
- *enabled*[4] (enabledness), i.e., is the activity available for execution in the current state of the workflow?

The only missing ingredient in DCR graphs is a way to express the value entered in fields. However, this has been implemented in recent (unpublished) extensions of DCR graphs[5], where the "executed" state has been combined with a data value, provided on each execution of the activity.

We shall in subsequent sections show how DCR graphs, *qua* these attributes, become a natural vehicle for expressing forms declaratively.

[4] "Enabledness" of a DCR activity is technically derived from the other three attributes.

[5] A related but different kind of DCR graphs with data appeared in [2].

3 Action taken

Exformatics A/S is qua DCRgraphs.net already a front-running vendor of declarative process modelling and simulation tooling. Observing the extent to which form semantics resemble declarative process models, we have defined and implemented a DCR-interpreter which presents a DCR graph as a form, following the principles laid out in Section 2. We shall see how this helps Exformatics and its customers in the next section.

3.1 DCR graphs

In this Subsection, we recall DCR graphs. We shall not give the formal definitions, but rather attempt to give the reader a brief informal introduction, sufficient to support the following Sections where we use DCR graphs for specifying forms. Readers familiar with DCR graphs are invited to take a quick look at the one in Figure 3 before skipping ahead to the next section. Readers interested in a more thorough introduction to DCR graphs are invited to look at either the informal introductions in one of [7,5], or the formal definitions in in [13,20,14,6,5].

The present development ascribes data values to activities and data guards to relations; this is an extension to the syntax and semantics of DCR graphs which has not yet appeared in the scientific literature.

A DCR comprises (1) a set of activities and (2) a set of relations between the activities. Activities are there to be executed, and relations indicate what changes happen to the state of the DCR graph as activities are executed. By convention, executing an activity in a DCR graph may input a *data value* for the activity.

As the name suggests a DCR graph is a graph: the nodes are activities, and the edges relations.

As a running example, we will use the DCR graph depicted in Figure 3. This DCR graph is a minimal model of the travel reimbursement workflow sketched in Section 2. It has just two activities (boxes): Approve or reject (to the left) and Justify Rejection (to the right).

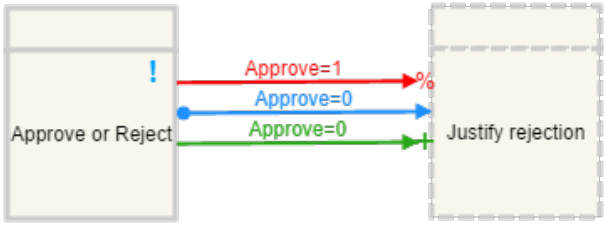

Fig. 3. Example DCR graph (reimbursement workflow)

As mentioned, each activity is at any given point in time in a particular *state*. This state comprises the attributes *included*, *pending*, an *executed*, as explained in Section 2 on page 4. Moreover, the activity may carry a data value, usually named something like the activity itself; and from the DCR graph and the attributes of an activity one derives whether or not the activity is *enabled*, i.e, currently able to execute.

The state of an activity is depicted graphically when drawing it as a box: The Justify rejection activity is *not included*, indicated by the dashed border of the box. On the other hand, the Approve or reject activity is *pending*, indicated by the blue exclamation mark.

As mentioned earlier, if any activity is pending *and included*, the workflow is incomplete; we say that the DCR graph is "not accepting". So because Approve or reject is pending, this workflow is not accepting.

DCR graphs have five relations; however, this paper will only discuss four. Three of these are exhibited in the example DCR graph in Figure 3, depicted as blue, red, and green arrows. Besides the colours, the arrows also have heads and tails to help distinguish them from each other.

The blue response *arrow (circle at the tail),* indicates that if the activity at the tail of the arrow is executed, the activity at its head is marked pending. The blue arrow in Figure 3 carries a *data guard*, indicating that the arrow takes effect only when the data expression in the guard is true. The expression in the guard may refer to the data value of the activity at its tail.

In the example, the blue arrow indicates that *if* Approve or reject is executed *and* with variable Approve having a value of 0/false (meaning that reject was selected), *then* Justify rejection is marked pending.

The green include *arrow (plus at the head) resp. the red* exclude *arrow (minus at the head)* indicate that when the activity at the tail is executed, the activity at the head is included resp. excluded. Excluded activities "do not count": They cannot be executed, and even if they are pending, they do not cause the workflow to be not accepting.

As with responses, include- and exclude-arrows may be guarded by data conditions. In this case, we see that when Approve or reject is executed, Justify rejection is either included or excluded, depending on whether the decision was to reject or approve.

The orange condition *arrow (circle at the head)* is not exhibited in this diagram; we shall see an example use of it later. It means that the activity head is prohibited from executing as long as the activity at the tail has not been executed or been excluded. Figure 5 has a condition between the Fill out expense report and Form, which mean that the Form cannot open before the Employee has filled out the expense report.

Note the subtle interplay of relations here: The blue response arrow on its own is not enough to force the manager to justify his rejection, because if Justify

rejection is excluded, it may be marked pending, but even so will not prevent the graph from being accepting. However, once we have also the green include arrow, Justify rejection is *both* marked pending *and* included, whence Justify rejection must now be executed or excluded for the workflow to be accepting.

This concludes our short tour of DCR graphs.

3.2 Forms from DCR graphs

In the previous sections we have introduced a simple travel reimbursement claim workflow, argued that it is tightly connected with the corresponding form, and seen how this workflow can be formalised as a DCR graph. We shall now see how to lift the workflow to be a form, following the principles laid out in Section 2.

DCR activities should be thought of formally as *events*; they represent things that happen, in principle without duration. However, as described above, the graph has a state, referred to as the marking. With the extension of DCR graphs where each execution of an activity is combined with a value stored as the current value of the activity, the marking provides a state of each activity that correspond closely to the state of fields in a form, where the value currently in the field is the value currently carried by the activity. That is, we consider the activity executed whenever the user updates the field, and we update the value of the activity with the value of the field whenever such an update happens.

With this connection, the connection sketched in Section 2 between state of a DCR activity on the one hand and UI-state of a form field on the other straightforwardly dictates the presentation of the DCR graphs as a form:

- Included activities are visible, excluded activities invisible.
- Pending activities must be executed or excluded before the form can be submitted[6]. I.e., the form cannot be submitted when a field is pending.
- Executed activities correspond to a field given a value.
- Enabled activities are read-write, disabled activities are read-only.

3.3 Visibility

We have already, if perhaps a bit implicitly, seen how included-state in the DCR graph becomes visibility in the form: The Justify Rejection field is not present initially (Figure 2), but appears once the claim is rejected (Figure 1).

Following this idea, and using the nesting construct of DCRgraphs, entire sections of form fields can be included/excluded (be visible/invisible) by the application of a single pair of include/exclude .

3.4 Mandatory fields: State vs. event

However, tension remains between the event-based nature of the acceptance criteria of DCR graphs and the state-based acceptance criteria of forms [15].

[6] In technical terms, the DCR graph must be *accepting* before the graph can be submitted [13].

The usual understanding of required fields in forms is that upon submit, that field must contain valid data. However, the notion of pending does not speak of the *state* of fields "at submit", just of that the *event* of filling a value in the field must happen before submission. If data can be invalidated, marking an event pending does of course not, make it necessary for the field to contain valid data at submission; it just means that at some point between now and submission, the user must have put data into it. In particular, the user is free to subsequently erase or invalidate the contents of the field.

To alleviate this tension, we must model more faithfully the dynamic behaviour of the form, taking into account the event of invalidating the required fields. We do so by adding to our model a response arrow from the pending activity to itself, guarded by value = null. This means that whenever the field is updated with a null value, it will be marked as pending again, ensuring that the workflow is not, in fact, complete before the activity/field is either excluded or receives a non-null value. Of course, one could define more complex criteria for when a value is invalid and thus trigger the response. In particular, the guard could depend on the value entered in other fields.

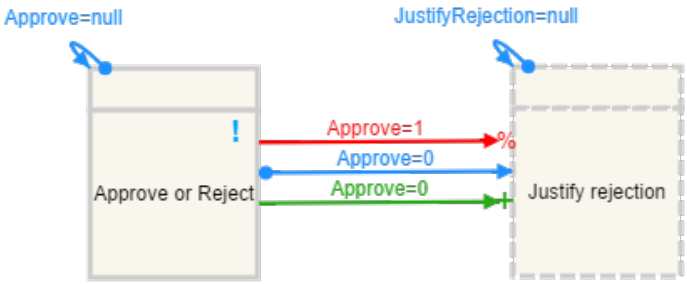

Fig. 4. Variant of DCR graph in Figure 3 taking null-values into account.

3.5 Workflow semantics of form execution

We have seen in the preceding subsections how a declaratively specified workflow *is* a specification of a form for inputting data into that workflow.

However, the classical expected semantics of a form is that no action was taken until the user clicks "submit". This is at odds with our interpreting activities as fields, and input is the field as execution of the activity. Taken as-is, this approach would mean that the moment a user inputs data into a field, the corresponding activity executes, potentially affecting the remainder of the workflow.

Conversely, execution of activities outside the form-activities may affect the included, enabled, and pending state of form activities, that is, the visibility,

writability, and requiredness of form fields. In the example in Figure 4 this cannot happen (because there are no activities not in the form), but in larger graphs, it could easily happen. E.g., an external audit might force additional data points to be required, in the DCR description including activities, which in the form becomes certain fields becoming visible.

Fields appearing spontaneously in a form, without user action, would be confusing and non-standard. This is unacceptable.

To address these two concerns, obtaining proper "submit" semantics and to avoid forms changing under the feet of the user, we treat forms as *transactions* (e.g., [1, Chapter 16]): Filling out the fields of a form does not, in fact, interact with the workflow:

- As the user interacts with the form, we collect the sequence of DCR activities executed.
- When the user clicks "submit", we attempt to replay that sequence of activity executions against the current state of the workflow.
- If this replay is successful, we update the state of the workflow to the resulting state from the replay.
- If the replay is not successful, e.g., because the user filled-in a field which has in the meantime become read-only (not enabled) or invisible (excluded), we display an error message to the user and *do not* update the state of the workflow.

The final version of the reimbursement claim workflow is as mentioned in Figure 4. This graph defines the form exhibited in Figures 2 and 1, with dynamic behaviour as specified in the bullet-list on page 4. In fact, those figures are screen-captures of forms automatically executed *directly from the DCR graph of Figure 4* by the DCRGraphs.net modelling and simulation tool. We invite the reader to experiment with the DCR graph of Figure 4 and its expression as a form in the DCRgraphs.net online tool. Please follow this link [9].

3.6 Integration of form & workflow

In this section, we show how to integrate our form in a larger workflow. We embed the the reimbursement approval form of Figure 4 in the following simple process:

- Employee fills out expense report
- Manager approves or rejects the claim (using the form)
- Finance reimburses the expenses

This process is depicted as a DCR graph in Figure 5.

DCRGraphs.net provides simulation features that also include DCR Forms. When a form button is enabled the event is displayed as Open rather than Execute, as we do not Execute the form until we Submit it successfully as described above.

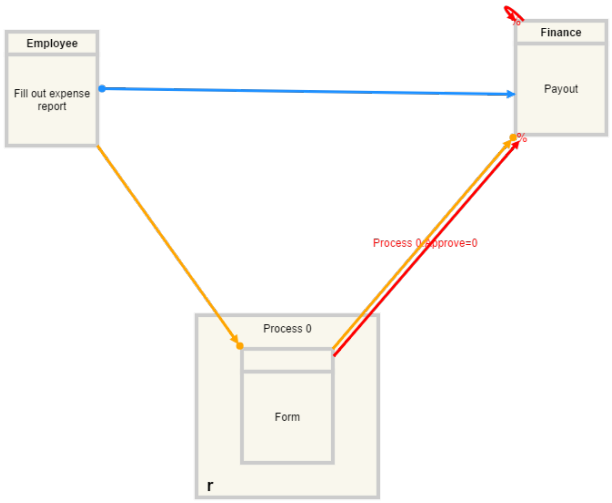

Fig. 5. Integration of form in larger process.

If we simulate the process and assume the Employee has filled out the expense report the window will look like Figure 6. Notice the task list. The notation Process 0 is a shorthand for opening the form event. Once the user clicks Open button, the form will be opened for user input.

Imagine the Manager rejecting the claim and given a reason hes ready to Submit the form (see Figure 7). Notice the exclude error that is guarded with Process 0.Approve=0. If the user choose to reject in the form the exclude relation will ensure Payout cannot happen.

4 Results achieved

Solutions sold by Exformatics invariably include forms as an important, sometimes primary, user interaction mechanism. As mentioned in the introduction, implementing a form is no different from implementing larger IT systems, and in Exformatics' experience, hamstrung by the same difficulties:

1. Customers never get the specification right the first time. E.g., when Exformatics in 2013 implemented an ACM solution for a Danish foundation based on DCR Graphs [23,3,4], they required more than 10 rounds of implementing successively closer-to-adequate form solutions before the customer finally had the solution they needed.
2. Implementing forms is time-consuming and expensive.
3. Even after the initial implementation, customers invariably require changes of various magnitudes, re-inducing the cost in time and space of (1) and (2).

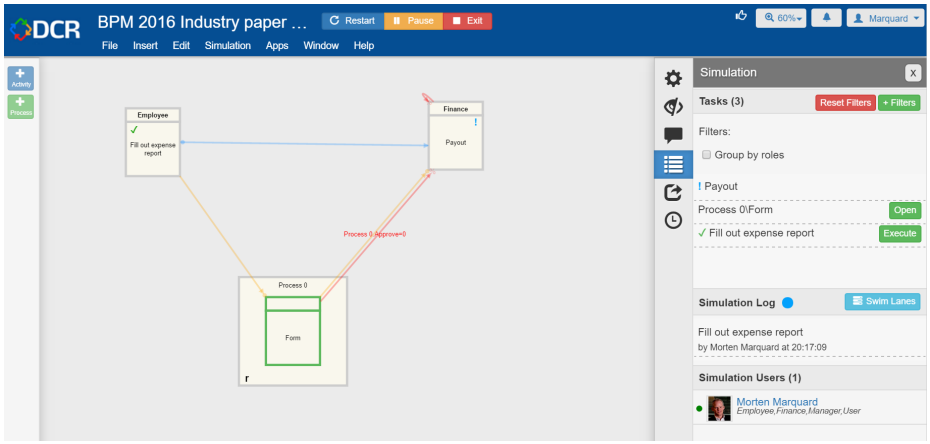

Fig. 6. Simulation of form (1)

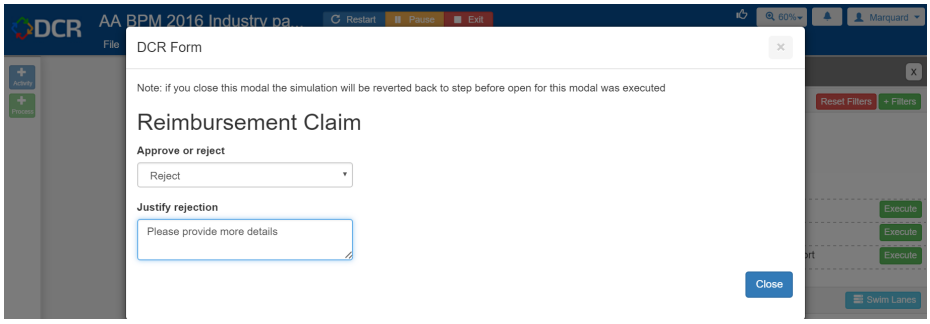

Fig. 7. Simulation of form (2)

Exformatics expects the present work to radically improve this situation, by building future Exformatics ACM solutions on forms based on DCR Graphs. The DCRGraphs.net online modelling and simulation tool is instrumental in this strategy, in that it addresses all of the above three concerns:

1. Workflows, hence forms, can be quickly and easily modelled in DCRgraphs.net. Moreover, DCRgraphs.net can simulate workflows *including processing forms* specified as part of the workflow using the approach of the present paper. This means that customers can immediately see proposed workflow and form solutions, and their feedback can be integrated in the proposed solution *on the spot*.
2. There is no implementation beyond formalising the form as a declarative workflow.
3. Changes in the form are reduced to changes in the workflow; declarative workflows and in particular DCR graphs have been demonstrated to be easy to adapt [6,5,3].

We emphasise that we expect the main benefit for both Exformatics and their customers is the ability to simulate forms in the tool immediately, avoiding confusion of behaviour as the form can be tested by users and adjusted as needed until it is correct. We expect a similar number of iterations to get the DCR Form correct, with significantly less time spend on each iteration.

Adopting new and changed business requirements will be easier as forms can be changed on the fly. A long term perspective is also the ability to allow run-time changes to forms. Imagine a case worker that needs to ask a client a set of information in a standard form. However, in the specific situation some additional information is needed. The case worker can simply adjust the specific instance of the form send to the client to accommodate the new information requirements.

Exformatics is currently implementing DCR Forms at Dreyers Fond because they need a minor change in the form, due to the European General Data Protection Regulation [21]. The ability to modify the form on the fly and simulate the changes immediately with the end-users makes it remarkably easy to agree on a new design with the customer and document the changes immediately.

The existing application form at Dreyers Fond is shown in Figure 8 and has been converted into DCR Forms as shown in Figure 9.

DCR Forms is currently in beta and we expect to change the existing legacy form at Dreyers Fond to the new DCR Forms based form within a few months.

5 Lessons learned

In this paper, we have demonstrated how the well-known UI-concept of "forms" can be fully specified as a declarative workflow. We have exemplified this with a small travel-reimbursement workflow, using the particular declarative notation of DCR graphs. The approach has been implemented by Exformatics A/S,

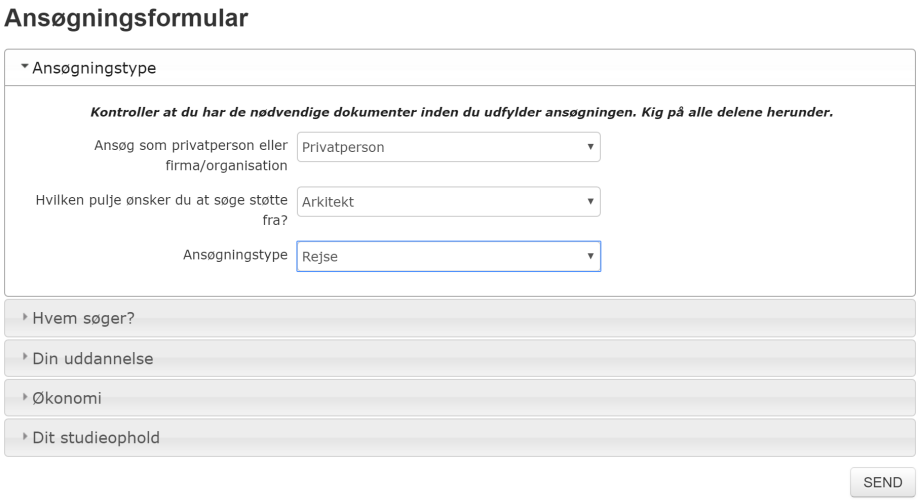

Fig. 8. The existing (legacy) application form at Dreyers Fond

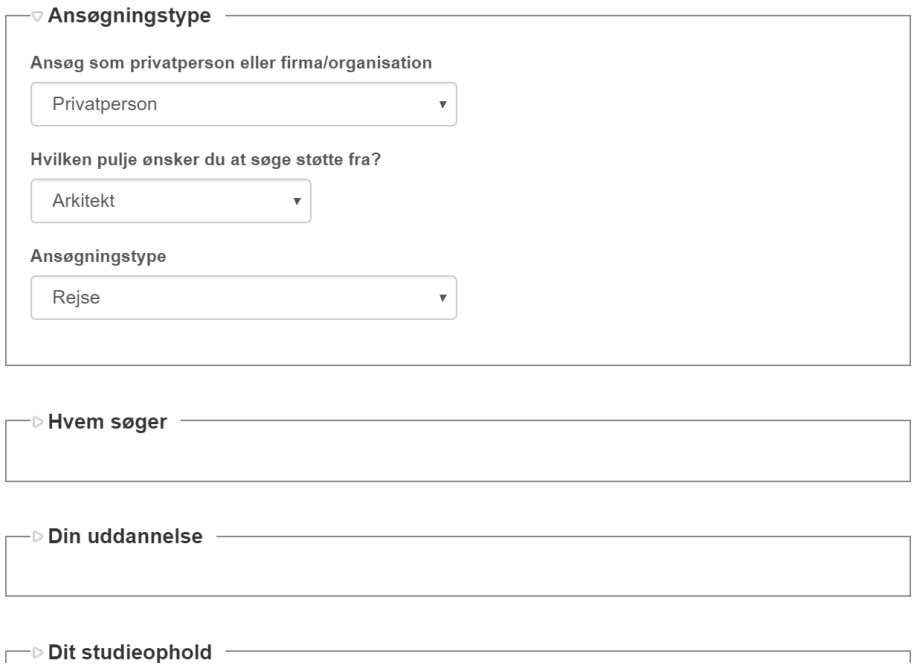

Fig. 9. The future application form at Dreyers Fond based on DCR Forms

and is available in their on-line process modelling and simulation tool DCR-graphs.net [10].

We emphasise that the link between form semantics and the rules governing the surrounding workflow are not only preserved, they are explicitly represented in this approach. That is, we obviate the distinction: The approach begins with a declarative workflow, and extracts from that the corresponding form, subject to minor changes to the workflow.

We see several exciting avenues of future work. In the short term, we intend to further alleviate the tension between the state-based nature of forms, and the event-based nature of DCR graph. In particular, we want to investigate how to combine the event-based constraints in DCR with primitives for expressing state-based constraints and acceptance criteria. In this study it will be relevant to look into data-centric models for case management (e.g. [16,17,18]) and the work on the Plato compiler [15], which generates web forms from declarative descriptions, but focuses only on data constraints.

In the longer run, letting declarative workflows specify forms opens up for the possibility of transferring other desirable properties from workflows to forms. In particular, DCR graphs allow run-time adaptations in various forms [6,3,19]. One exciting opportunity afforded by the present approach is to allow case workers to augment forms on a case-by-case basis: A caseworker may decide that in this *particular* instance, a client must provide an additional data-point. If the case-worker has access to modifying the declaratively specified workflow, she can simply *add the corresponding activity.*

References

1. Coulouris, G., Dollimore, J., Kindberg, T., Blair, G.: Distributed Systems: Concepts and Design. Addison-Wesley Publishing Company, USA, 5th edn. (2011)
2. Debois, S., Hildebrandt, T.: Dynamic condition response graphs as foundation for event-based languages and systems (2015)
3. Debois, S., Slaats, T.: The analysis of a real life declarative process. In: CIDM 2015. pp. 1374–1382 (2015)
4. Debois, S., Hildebrandt, T., Marquard, M., Slaats, T.a.: A case for declarative process modelling: Agile development of a grant application system. In: International Workshop on Adaptive Case Management and other non-workflow approaches to BPM (2014)
5. Debois, S., Hildebrandt, T., Slaats, T.: Hierarchical Declarative Modelling with Refinement and Sub-processes. In: Business Process Management, Lecture Notes in Computer Science, vol. 8659, pp. 18–33. Springer International Publishing (2014), http://dx.doi.org/10.1007/978-3-319-10172-9_2
6. Debois, S., Hildebrandt, T., Slaats, T.: Safety, liveness and run-time refinement for modular process-aware information systems with dynamic sub processes. In: Bjørner, N., de Boer, F. (eds.) FM 2015: Formal Methods: 20th International Symposium, Oslo, Norway, June 24-26, 2015, Proceedings. pp. 143–160. Springer International Publishing, Cham (2015), http://dx.doi.org/10.1007/978-3-319-19249-9_10

7. Debois, S., Hildebrandt, T.T., Marquard, M., Slaats, T.: Hybrid process technologies in the financial sector. In: BPM '15 (Industry track). pp. 107–119 (2015), http://ceur-ws.org/Vol-1439/paper9.pdf
8. Debois, S., Hildebrandt, T.T., Slaats, T.: Concurrency and Asynchrony in Declarative Workflows. In: Motahari-Nezhad, H.R., Recker, J., Weidlich, M. (eds.) Business Process Management - 13th International Conference, BPM 2015, Innsbruck, Austria, August 31 - September 3, 2015, Proceedings. Lecture Notes in Computer Science, vol. 9253, pp. 72–89. Springer (2015)
9. Exformatics A/S: Dcrgraphs editor and simulator, form of figure 4 (2016), http://www.dcrgraphs.net/Tool?id=2398
10. Exformatics A/S: DCRGraphs editor and simulator, DCRGraphs.net (2016), DCRGraphs.net
11. Harms, J.: Research Goals for Evolving the Form User Interface Metaphor towards More Interactivity. In: Holzinger, A., Ziefle, M., Hitz, M., Debevc, M. (eds.) Human Factors in Computing and Informatics, pp. 819–822. No. 7946 in Lecture Notes in Computer Science, Springer Berlin Heidelberg (2013), http://link.springer.com/chapter/10.1007/978-3-642-39062-3_60, dOI: 10.1007/978-3-642-39062-3_60
12. Harms, J.: Past, present, and future of form-based user interfaces: Innovative design for evolving the 'form' user interface metaphor. PhD thesis supervised by T. Grechenig. Vienna University of Technology (2016)
13. Hildebrandt, T., Mukkamala, R.R.: Declarative event-based workflow as distributed dynamic condition response graphs. In: Post-proc. of PLACES 2010 (2010), http://www.itu.dk/~rao/pubs_submitted/dcrsjournalversionfinal.pdf
14. Hildebrandt, T.T., Mukkamala, R.R., Slaats, T.: Nested Dynamic Condition Response Graphs. In: Proc. of the 4th Int. Conf. on Fundamentals of Software Engineering, FSEN. pp. 343–350 (2011)
15. Hinrichs, T.L.: Plato: A Compiler for Interactive Web Forms. In: Rocha, R., Launchbury, J. (eds.) PADL, pp. 54–68. No. 6539 in LNCS, Springer Berlin Heidelberg (Jan 2011), dOI: 10.1007/978-3-642-18378-2_7
16. Hull, R., et.al.: Introducing the guard-stage-milestone approach for specifying business entity lifecycles. In: Proc. of 7th Intl. Workshop on Web Services and Formal Methods (WS-FM 2010), Revised Selected Papers. Lecture Notes in Computer Science, Springer (2010)
17. Hull, R., et.al.: Business artifacts with guard-stage-milestone lifecycles: Managing artifact interactions with conditions and events. In: ACM Intl. Conf. on Distributed Event-based Systems (DEBS). pp. 51–62 (2011)
18. Kumaran, S., Nandi, P., III, F.T.H., Bhaskaran, K., Das, R.: ADoc-oriented programming. In: Symp. on Applications and the Internet (SAINT). pp. 334–343 (2003)
19. Marquard, M., Shahzad, M., Slaats, T.: Web-based modelling and collaborative simulation of declarative processes. In: BPM 2015. pp. 209–225 (2015)
20. Mukkamala, R.R.: A Formal Model For Declarative Workflows: Dynamic Condition Response Graphs. Ph.D. thesis, IT University of Copenhagen (June 2012)
21. REGULATION (EU) 2016/679 OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation) (Apr 2016)

22. Seckler, M., Heinz, S., Bargas-Avila, J.A., Opwis, K., Tuch, A.N.: Designing usable web forms: empirical evaluation of web form improvement guidelines. pp. 1275–1284. ACM Press (2014), http://dl.acm.org/citation.cfm?doid=2556288.2557265
23. Slaats, T., Mukkamala, R., Hildebrandt, T., Marquard, M.: Exformatics declarative case management workflows as DCR graphs. In: BPM 2013 (2013)
24. Slaats, T.: Flexible Process Notations for Cross-organizational Case Management Systems. Ph.D. thesis, IT University of Copenhagen (Jan 2015)