The DCR Workbench: Declarative Choreographies for Collaborative Processes

Søren Debois and Thomas T. Hildebrandt

Department of Computer Science, IT University of Copenhagen, Rued Langgaards Vej 7, 2300 Copenhagen S, Denmark

Abstract

The DCR Workbench is an online tool for simulation and analysis of collaborative distributed processes specified as DCR graphs. The Workbench is a robust and comprehensive implementation of DCR graphs, providing concrete syntax, specification by refinement, visualisation, simulation, static analysis, time analysis, enforcement, declarative subprocesses, data dependencies, translation to other declarative models, and more. This chapter introduces the Workbench and, through the features of the Workbench, surveys the DCR formalism. The Workbench is available on-line at http://dcr.tools.

5.1 Introduction

Citizens, businesses and public organisations increasingly rely on distributed business processes. Many such processes involve at the same time information systems, humans and mechanical artefacts, and are thus highly unpredictable. Moreover, such processes are constantly evolving due to advances in technology, improvement in business practices, and changes in legislation.

In this climate of distribution and continuous change, the traditional vision of verifying a system once and for all against a final formal description has little hope of realisation. Instead, we need tools and techniques for describing, building, and analysing systems of continuously changing distributed collaborative processes. Dynamic Condition Response graphs, DCR graphs, is a formal model developed in response to this need.
The DCR Workbench: Declarative Choreographies for Collaborative Processes

Developed through a series of research projects, DCR graphs today stands on three pillars: a substantial body of academic publications on both case studies [8, 10, 15, 17, 20, 38] and formal aspects [2, 6, 7, 12–14, 16, 18, 21–25, 29–31, 34, 37]; the DCR Workbench, implementing most major advances of the formalism (the subject of this chapter); and a commercial adaptive case-management system developed by independent vendor Exformatics A/S [9, 11, 15, 19, 23, 28].

Declarative process notations such as DCR graphs, DECLARE [33, 40] and GSM [26] generally support specification and analysis of requirements, whereas imperative notations such as Workflow Nets [1] and BPMN [32] generally support implementation of requirements. DCR graphs have the advantage of serving as both the specification of requirements and the runtime representation of a process instance, which can be adapted dynamically if the requirements change.

The DCR Workbench is a comprehensive tool for modelling with DCR graphs and analysing DCR models. The Workbench serves the dual purposes of being a communication and teaching tool, used both in classroom settings and in discussions with industry, as well as a test-bed for experimentation with new analysis and variants.

This chapter gives an introduction to DCR graphs in general and the Workbench in particular. As we shall see, the Workbench implements a majority of published DCR graph variants and analysis methods, as well as some work-in-progress experimental additions and algorithms that have yet to be published. Through the features of the Workbench, the chapter also provides a survey of the state-of-the-art of DCR graphs variants, their technical properties, and their published analysis methods and algorithms.

5.1.1 History of the DCR Workbench

DCR graphs were introduced in 2010 [18, 29] by Thomas Hildebrandt and his group at the ITU. Soon after, Danish vendor of adaptive case-management solutions, Exformatics A/S, entered into a long-term collaboration with the ITU group; a collaboration which continues to this day. The continued financial support and interest of Exformatics A/S has been instrumental in the development of the formalism.

DCR graphs were implemented repeatedly as the formalism evolved. Notably, an early implementation created by industrial PhD Tijs Slaats in collaboration with Exformatics A/S [37] eventually grew into that company’s current commercial DCR tool [9, 15, 28], available at dcrgraphs.net.
2013 this tool was solidifying into a commercial offering. While the backing of commercial vendor was extremely helpful to DCR graph, the Exformatics tool was becoming too heavyweight for quick academic experiments. Accordingly, the ITU group in 2013 commenced development of a nimbler implementation. This effort was spearheaded by Søren Debois and became the DCR Workbench of the present chapter.

The two tools have different goals: Exformatics’ offering is aimed at non-expert commercial users and emphasises stability and usability. Conversely, the DCR Workbench is aimed at academics and prioritises ease-of-experimentation overall. This division has so far been productive: sufficiently good ideas implemented in the Workbench has later been re-implemented by Exformatics in their commercial offering [8–10, 28].

The Workbench made its first appearance in a research paper in 2014 [12], and its first appearance in industry collaborations in 2015 [9]. Subsequently, the Workbench has provided implementation and examples for most major developments of the formalism [2, 6, 12–14, 16].

The DCR Workbench is implemented in F# [39], using the WebSharper library [5] to derive server- and client-side components from the same F# code base. The choice of implementation language and platform is no accident: On the one hand, F# is very well-suited to manipulating formal models; on the other, the web-based platform makes the Workbench immediately available to interested researchers: all it takes is a browser.

### 5.1.2 The DCR Workbench

The DCR Workbench is available at

http://dcr.tools/2017chapter

This URL leads to a special page supporting this chapter with the collection of examples used on the following pages. We encourage the reader to visit this page and actively try out the examples presented in the remainder of this chapter as he progresses through the it.

**Overview** In Section 5.2, we introduce a running example, and in Section 5.3, we recall DCR graphs. In Section 5.4 we introduce basic modelling, simulation and analysis of DCR graphs in the Workbench. In Section 5.5 we construct models by refinement; in Section 5.6 we discuss timed models; in Section 5.7 we talk about subprocesses; and in Section 5.8 data. In Section 5.9, we mention briefly other tools in the Workbench, before concluding in Section 5.10.
5.2 Running Example

As a running example we consider a stylised mortgage loan application process distilled from real-life cases [9, 13]. Mortgage application processes are in practice extremely varied, depending on the type of mortgage, the neighbourhood, the applicant, and the credit institution in question. The purpose of the process is to arrive at a point where the activity Assess loan application can be carried out. This requires in turn:

1. collecting appropriate documentation,
2. collecting a budget from the applicant, and
3. appraising the property.

In practice, applicants’ budgets tend to be underspecified, so an intern will screen the budget and request a new one if the submitted one happens to be so. The case worker should not spend time assessing the application if the documentation has not been collected or the budget is underspecified. The caseworker decides if the appraisal can be entirely statistical, i.e., carried out without physical inspection, or if it requires an on-site appraisal. For reasons of cost efficiency, only one appraisal should be carried out.

5.3 Dynamic Condition-Response Graphs

In this section, we recall DCR graphs [6, 9, 14, 15, 18, 29, 37]. We begin by an informal walkthrough, followed by a formal development in Section 5.3.4.

DCR graphs constitute a declarative modelling notation describing at the same time a process and its run-time state. The core notation comprises labelled events, event states, and five possible relations between events. The relations govern: (a) how executability of one event depend on the state of another, and (b) how execution of one event updates the states of another.

5.3.1 Event States

The event state consists of three booleans: The executed, included, and pending states of the event.

- The executed state simply registers whether the event has been previously executed (an event may execute more than once). It is updated to true whenever the event executes. It is never updated to false.
- The included state indicates whether the event is included, i.e. relevant for the process. Being included is a prerequisite for an event to execute.
5.3 Dynamic Condition-Response Graphs

- The pending state indicates whether the event is required to eventually execute (or become not included).

We give events and initial states for the running example in Figure 5.1. Except Request new budget, which becomes relevant only when a budget has been submitted, all events are included. The Assess loan application and Submit budget events are pending: they are required to complete the process.

5.3.2 Relations

Each pair of events may be related by one of five different relations. Relations regulate (a) which events may execute in a given graph (condition, milestone) and (b) the effect of executing an event (inclusion, exclusion, response).

We give a full DCR model of the running example\(^1\) in Figure 5.2.

**Conditions.** A condition \(e \rightarrow \bullet f\) causes the target activity \(f\) to be not executable whenever the source activity \(e\) is included (its “included” state is true) and has not been previously executed (its “executed” state is false). E.g., in Figure 5.2, we must execute Collect documents before Assess loan application can be executed.

**Milestones.** A milestone \(e \rightarrow \diamond f\) causes the target activity \(f\) to be not executable whenever the source activity \(e\) is included and pending (its “included” and “pending” states are true). In Figure 5.2, whenever Submit budget is pending, Assess loan application is prevented from executing.

<table>
<thead>
<tr>
<th>Event</th>
<th>Role</th>
<th>Initial state</th>
</tr>
</thead>
<tbody>
<tr>
<td>Collect documents</td>
<td>Caseworker</td>
<td></td>
</tr>
<tr>
<td>Budget screening approve</td>
<td>Intern</td>
<td></td>
</tr>
<tr>
<td>Request new budget</td>
<td>Intern</td>
<td>Excluded</td>
</tr>
<tr>
<td>Submit budget</td>
<td>Customer</td>
<td>Pending</td>
</tr>
<tr>
<td>On-site appraisal</td>
<td>Mobile consultant</td>
<td></td>
</tr>
<tr>
<td>Statistical appraisal</td>
<td>Caseworker</td>
<td></td>
</tr>
<tr>
<td>Assess loan application</td>
<td>Caseworker</td>
<td>Pending</td>
</tr>
</tbody>
</table>

Figure 5.1 Events and initial states (marking) for the mortgage application process. Where nothing else is indicated, the initial state of an event is not executed, included, and not pending.

\(^1\)This graph is in fact the output of the DCR Workbench visualiser; we describe in Section 5.4.2 exactly how the visualiser represents event state.
Responses. A response $e \rightarrow f$ causes the target activity $f$ to be pending (its “pending” state is true) whenever the source activity $e$ is executed. In Figure 5.2, when an applicant executes Submit budget, we require a subsequent screening: there is a response from Submit budget to Budget screening approve.

Inclusions and exclusions. An inclusion $e \rightarrow+ f$ resp. an exclusion $e \rightarrow% f$ causes the “included” state of the target activity $f$ to be true resp. false when the source activity $e$ is executed. In Figure 5.2, the activity Submit budget includes the activity Request new budget.

### 5.3.3 Executing Events

Enabled events are the ones which have their “included” state true, and which do not have their execution prohibited by a condition or milestone as indicated above. Conditions or milestones from excluded events do not disable their target events. E.g., in Figure 5.2, both On-site appraisal and Statistical appraisal are conditions for Assess loan application, but once one executes, the other is excluded and thus no longer required for Assess loan application to execute.
5.3 Dynamic Condition-Response Graphs

Executing an enabled event in a DCR graph updates event states as indicated by inclusion, exclusion, and response relations. Conceptually, we can think of the execution as producing a new DCR graph with the same relations but with updated event states.

The denotation of a DCR model is the set of (finite and infinite) sequences of event labels, corresponding to a sequence of such event executions, where at each step, every pending event is eventually executed or excluded at some later step. We refer to such sequences as accepting traces. It follows that for finite accepting traces, in the final state, no event is pending and included.

We consider potential traces for Figure 5.2.

- $s_0 = \langle \text{Collect documents}, \text{Assess loan application} \rangle$. This sequence intuitively corresponds to assessing the loan application without getting a budget and appraising the property. After Collect documents, the event Assess loan application is not enabled, so $s_0$ is not a trace.
- $s_1 = \langle \text{Collect documents}, \text{Submit budget} \rangle$. This sequence is a trace, but not an accepting one, since Assess loan application is pending and included in the final state. It follows that $s_1$ sequence is not part of the denotation of Figure 5.2.
- $s_2 = \langle \text{Collect documents}, \text{Submit budget}, \text{Budget screening approve}, \text{Statistical appraisal}, \text{Assess loan application} \rangle$ is an accepting trace.

Notice that between the notions of enabledness and accepting trace, DCR graphs express both permissions and obligations. We return to expressiveness of DCR graphs in Section 5.7 below.

5.3.4 Formal Development

**Definition 1 (DCR Graph [18]).** A DCR graph, ranged over by $G$, is a tuple $(E, R, M, \ell)$ where

- $E$ is a finite set of (labelled) events, the nodes of the graph.
- $R$ is the edges of the graph. Edges are partitioned into five kinds, named and drawn as follows: The conditions ($\rightarrow \bullet$), responses ($\bullet \rightarrow$), milestones ($\rightarrow \diamond$), inclusions ($\rightarrow +$), and exclusions ($\rightarrow \%$).
- $M$ is the marking of the graph. This is a triple $(Ex, Re, In)$ of sets of events, respectively the previously executed (Ex), the currently pending (Re), and the currently included (In) events.
- $\ell$ is a labelling function assigning to each $e \in E$ a label comprising an activity name and a set of roles.
When $G$ is a DCR graph, we write, e.g., $E(G)$ for the set of events of $G$, $Ex(G)$ for the executed events in the marking of $G$, etc.

**Notation.** Let $R \subseteq X \times Y$ be a relation. For $y \in Y$ we take $R y = \{ x \in X \mid (x, y) \in R \}$; dually for $x \in X$ we take $x R = \{ y \in Y \mid (x, y) \in R \}$. We use this notation for relations, e.g., $(\rightarrow \bullet e)$ is the set of events that are conditions for $e$.

**Definition 2 (Enabled events).** Let $G = (E, R, M, \ell)$ be a DCR graph, with marking $M = (Ex, Re, In)$. An event $e \in E$ is enabled, written $e \in \text{enabled}(G)$, iff (a) $e \in In$, (b) $In \cap (\rightarrow \bullet e) \subseteq Ex$, and (c) $In \cap (\rightarrow \diamond e) \subseteq E \setminus Re$.

That is, enabled events (a) are included, (b) have their included conditions executed, and (c) have no included milestone with an unfulfilled responses.

**Definition 3 (Execution).** Let $G = (E, R, M, \ell)$ be a DCR graph with marking $M = (Ex, Re, In)$. Suppose $e \in \text{enabled}(G)$. We may execute $e$ obtaining the DCR graph $G' = (E, R, M', \ell)$ with $M' = (Ex', Re', In')$ defined as follows.

1. $Ex' = Ex \cup e$
2. $Re' = (Re \setminus e) \cup (e \rightarrow \bullet)$
3. $In' = (In \setminus (e \rightarrow \%)) \cup (e \rightarrow +)$

That is, to execute an event $e$ one must: (1) add $e$ to the set $Ex$ of executed events; (2) update the currently required responses $Re$ by first removing $e$, then adding any responses required by $e$; and (3) remove from $In$ those events excluded by $e$, then adding those included by $e$.

Technically, the operational semantics of a DCR graph is the labelled transition system where states are graphs and transitions are executions.

**Definition 4 (Transitions).** Let $G$ be a DCR graph. If $e \in \text{enabled}(G)$ and executing $e$ in $G$ yields $H$, we say that $G$ has transition on $e$ to $H$ and write $G \rightarrow e H$. A run of $G$ is a (finite or infinite) sequence of DCR graphs $G_i$ and events $e_i$ such that: $G = G_0 \rightarrow e_0 G_1 \rightarrow e_1 \ldots$ A trace of $G$ is a sequence of labels of events $e_i$ associated with a run of $G$. We write $\text{runs}(G)$ and $\text{traces}(G)$ for the set of runs and traces of $G$, respectively.

The denotation of a DCR graph is the set of accepting finite and infinite traces allowed by its operational semantics.

**Definition 5 (Acceptance).** A run $G_0 \rightarrow e_0 G_1 \rightarrow e_1 \ldots$ is accepting iff for all $n$ with $e \in \text{In}(G_n) \cap \text{Re}(G_n)$ there exists $m > n$ s.t. either $e_m = e$, or $e \not\in \text{In}(G_m)$. A trace is accepting iff it has an underlying run which is.
5.4 Modelling with the Workbench

Acceptance tells us which workflows a DCR graph accepts, its \textit{language}.

\textbf{Definition 6 (Language).} The \textit{language} of a DCR graph $G$ is the set of its accepting traces. We write $\text{lang}(G)$ for the language of $G$.

We conclude this Section by noting that by Definitions 2 and 3, because the set of events is finite, both the set of enabled events and the result of executing an event are computable in polynomial time.

5.4 Modelling with the Workbench

A typical configuration of the Workbench can be seen in Figure 5.3. The Workbench is divided into panels. In the configuration in Figure 5.3, we see the Visualiser, Parser and Activity Log panels.

The Workbench maintains at all times a current DCR graph and a current trace. Each panel allow the user to interact with this current graph and current trace. A few panels also maintain a DCR graph of their own.

Panels are dynamic: The user is free to remove panels by clicking “close” in the lower-left corner of a panel; or to add panels by selecting a new panel in the “Add a new panel” section of the Workbench panel. At the time of writing, the Workbench implements 22 different panels.

When working with the Workbench, it is customary to have several panels open; e.g., a visualiser and one or more analysis panels. The Workbench panel contains a selection of seven pre-made such panel configurations called “presets”. These presets are accessible through the left-hand “Load a preset” section of the Workbench panel.

Finally, the Workbench can function as a process engine, making some of its functionality available programmatically as a REST interface; see the right-hand “REST API” section.

5.4.1 Inputting a Model: The Parser Panel

The parser panel allows input of DCR graphs as plain text. The parser accepts programs written according to the grammar of Figure 5.4.

As an example program, consider the abridged variant of our running example given in Figure 5.6; the corresponding input program is listed in Figure 5.5. The Workbench accepts such source programs as input, producing visualisations automatically. (Visualisations in this chapter was so produced.)
Figure 5.3 The DCR Workbench (http://dcr.tools).
5.4 Modelling with the Workbench

Figure 5.4  EBNF definition of the language recognised by the Parser panel.

The concrete syntax specifies events and relations such as “condition from A to B” with expressions such as “A \( \rightarrow^{*} \) B”. An event state is specified by prefixing an event with modifiers such as ! or \%. We see this on line 6 in Figure 5.5. If the event occurs more than once in the program, it is sufficient to prefix the modifier only once. We specify roles by adding a role tag to the event. We see this on line 3 in Figure 5.5. More than one role may be added; in general, the same tag may be added multiple times.

It is occasionally convenient to relate more than one event at the same time. In the present case, Assess loan application needs conditions on both
Collect documents and Submit budget. We specify these conditions concisely by enclosing the latter two in parenthesis, as seen on line 3–4.

As the user types in the parser, a preview of the graph being input is presented on the right. The input graph substitutes the current graph and resets the current trace when the user clicks “Load”.

The parser also understands the XML format output by Exformatics A/S commercial http://dcrgraphs.net tool [15, 38].

5.4.2 Visualisation and Simulation: The Visualiser and Activity Log Panels

The Visualiser, top-most in Figure 5.3, displays a visualisation of the current DCR graph. In Figure 5.3, the chosen visualisation is simply the graph layout; alternatively, the underlying transition system may be shown (see below).

The visualisation of the core application process (Figure 5.5) is reproduced in Figure 5.6. The visualiser represent events as boxes, labelled by the activity of the event (centre) and the role or participant executing that activity (top). E.g., the top-left box represents an activity Collect documents which is carried out by a Caseworker.

Activities are coloured according to their state: grey background is not currently executable (Assess loan application in Figure 5.6), red label! with an exclamation mark is pending (ditto); “greyed out” boxes are excluded events (Request new budget in the original Figure 5.2); and finally, executed events have a tick mark after their action label, (Submit budget in Figure 5.3).
5.5 Refinement

**Simulation**  The visualiser allows executing events by clicking. E.g., to execute Submit budget, simply click it. This will extend the current trace with that execution, and replace the current graph with the one obtained by applying the updates to event state resulting from the execution of Submit budget (in this case, setting “executed” of that event true). Use the browser’s back buttons to revert to a previous state.

The Activity Log panel, third from the top in Figure 5.3, displays the current trace, analogous to the way the visualiser displays the current graph.

**State-space enumeration**  As mentioned in Definition 4, a DCR graph gives rise to a labelled transition system (LTS), where states are markings and transitions are labelled event executions. The visualiser can be configured to render a visualisation of the state space of the DCR graph rather than the DCR graph itself, through the drop-down button on the left of the panel. The visualiser highlights the current run in that LTS. The visualisation of the full LTS of the full mortgage application process of Figure 5.2.

The visualiser was originally reported in [12], with the transition system generator following in [9].

### 5.5 Refinement

We proceed to construct step-wise the full mortgage process application by refinement [6,14]. We begin with the core process of Figures 5.5 and 5.6. We first add the process fragments for budget submission and screening given in Figures 5.8 and 5.9.

The Workbench supports step-wise refinement: by using in the parser the “Merge” button rather than the “Load” button. Whereas “Load” replaces the current graph and sets the current trace to empty, the “Merge” button preserves both, replacing the global current graph with (graph) union $G \oplus H$ of the current graph $G$ and the parser’s current local graph $H$.

To refine the core process by the budget fragment, we make sure that the core process is the current graph, then enter the fragment (Figure 5.9) in the parser, and click “Merge”. The result is the graph in Figure 5.10. As can be seen, the resulting process is close to the full running example in Figure 5.2, except the process fragment for appraising the property is missing.
Repeating the Merge procedure with the process fragment in Figure 5.11 adds the missing bits and leaves us with exactly Figure 5.2—this is how the examples for the present chapter has been constructed.
Not every such merge preserves the language of the original graph. Exclusions may void conditions, giving the merged graph behaviour not present in the original graph, even when restricting attention to only the events of that graph. As a very simple example, consider the two graphs $G = a \rightarrow b$ and $H = c \rightarrow a$. The union $G \oplus H = a \rightarrow b, c \rightarrow a$ has the trace $\langle c, b \rangle$; even if we dismiss the new event $c$, $G$ could not by itself exhibit the trace $\langle b \rangle$. 

Figure 5.8  Budget process fragment.

Figure 5.9  Visualisation of the budget process fragment of Figure 5.8.
Visualisation of the core process (Figures 5.5 and 5.6) refined by the budget fragment (Figures 5.8 and 5.9).

Group "Appraisal" {
    "On-site appraisal" [ role = "Mobile consultant" ]
    "Statistical appraisal" [ role = "Caseworker" ]
}

"Statistical appraisal" -->% "On-site appraisal"
"On-site appraisal" -->% "Statistical appraisal"
"Appraisal"
-->* "Assess loan application" [ role = "Caseworker" ]

Appraisal process fragment.

This situation was investigated in detail in previous work [6, 14], where a sufficient condition for a refinement to be language preserving in the above sense was established in a much richer setting. For the present notion of DCR
graphs, it is sufficient to require that the refining graph does not exclude or include any events of the original graph.

**Notation.** Given a sequence $s$ and an alphabet $\Sigma$, write $s \upharpoonright \Sigma$ for the largest sub-sequence $s'$ of $s$ s.t. $s'_i \in \Sigma$; e.g., if $s = AABC$ then $s \upharpoonright_{A,C} = AAC$.

**Definition 7.** Given DCR graphs $G$ and $H$, we say that $H$ is a refinement of $G$ iff $\lang(H) \upharpoonright_{\ell(E(G))} \subseteq \lang(G)$.

That is, the language of $H$ restricted to the labels used by events in $G$ must be a subset of the language of $G$. We can now state the following Proposition [6, Theorem 43]:

**Proposition 1.** Let $G$ and $G'$ be DCR processes such that for every $e \in E(G)$, there is no relation $x \rightarrow e$ or $x \rightarrow e$ in $G'$. Then the graph union $G \oplus G'$ is a refinement of $G$.

If the Workbench has current graph $G$ and the parser has graph $G'$ not satisfying (a published [6,14] generalisation of) the conditions of Proposition 1, the Parser issues a warning and requires confirmation before merging.

DCR refinement was originally suggested in [17, 24] and worked out comprehensively in [6, 14]. The Workbench implements this latter mechanism.

## 5.6 Time

The Workbench supports the extension of DCR graphs with time [2, 23]. Time is modelled discretely by a special action tick modelling the passage of time; conditions are augmented with an optional delay, $e \xrightarrow{k}\cdot f$, and responses with an optional deadline $e \xrightarrow{} f$. Intuitively, the delay in the timed condition requires that at least $k$ ticks have passed after the last execution of $e$ before $f$ may execute; dually, the deadline in the timed response requires that at most $k$ ticks pass after the last execution of $e$ before $f$ must execute.

The former requirement makes it possible to have **timelocks**, i.e., situations where, say, $f$ must execute, but is not allowed to. As a very simple example, consider the DCR graph $G = e \xrightarrow{3}\cdot f, e \xrightarrow{2}\cdot f$. In this graph, once $e$ executes, *at least* 3 ticks must pass before $f$ can execute because of the condition delay, but *at most* 2 ticks may pass before $f$ must execute because of the response deadline. After the sequence $\langle e, \text{tick}, \text{tick} \rangle$, the graph is said to be **time-locked**: Time cannot advance without a constraint being violated.
For our running example, suppose that (a) the initial screening of the customer’s budget must be completed within 5 days, and (b) that the final assessment of the loan application must wait a 3-day “grace period” after a statistical appraisal (in order to prevent caseworkers from doing overly optimistic statistical appraisals). We model these constraints as a timed DCR graph directly using a timed response and condition in Figures 5.12 and 5.13.

Timed DCR graphs are still in finite state [23], but deciding time-lock freedom naively by exploring in the state space is infeasible. Recent research [2] established a sufficient condition for a graph to be time-lock–free and gave a generic “enforcement mechanism” for time-lock–free graphs, that is, a device which monitors the progression of time and a DCR graph and proactively causes events to execute to avoid missing deadlines.

The Workbench implements time as defined in [23], and time-lock–analysis and enforcement as defined in [2].

"Submit budget" [ role = "Customer" ]
*-[5]-> "Budget screening approve" [ role = "Intern" ]

"Statistical appraisal" [ role = "Caseworker" ]
-[3]->" Assess loan application" [ role = "Caseworker" ]

Figure 5.12 Additional timing constraints for the mortgage application process in Figure 5.2.
5.7 Subprocesses

It may happen that a customer during the application process applies for pre-approval of an expected increase in property value due to, e.g., ongoing kitchen remodellings. In this case, the caseworker must assess the limit extension before deciding on the mortgage application itself. At the caseworker’s discretion, an intern may or may not collect bank statements from the customer for the limit extension assessment; however, collecting that statement requires the customer’s explicit consent.

Such limit extensions in practice may happen several times during a mortgage application due to, e.g., expanded scope of a kitchen remodelling project. Thus, the limit extension fragment is a subprocess: A process that may be added to the main process when necessary, and possibly repeatedly.

Note that since subprocesses may be added repeatedly, each such addition must duplicate the events of the subprocess. This situation is akin to bound names under replication being duplicated in in the π-calculus [35]. The subprocess may contain both events local to the subprocess, bound events, and references to events of the containing graph. The former are indicated syntactically with a / prefix as seen in lines 6–8.

The Workbench supports subprocesses; we add the above limit extension process in Figures 5.14 and 5.15. Note that in visualisation, the subprocess is not visible until it has been expanded once.

The visualisation shows the triggering event Apply for limit extension—singled out as spawning a subprocess by the ⊞ following contemporary business process notations, e.g., BPMN [32]. The bound events in a subprocess are shown with round corners and inside a dashed box2.

"Assess loan application"

"Apply for limit extension" [ role = Customer ]
{ /*"Assess limit extension" [ role = Caseworker ]
   /*Collect consent" [ role = Intern ]
   -->* /*"Collect bank statement" [ role = Intern ]
   "Submit budget" [ role = Customer ]
   --> !"Assess limit extension" [ role = Caseworker ]
   -->* "Assess loan application" [ role = Caseworker ]
}
*--> "Submit budget"

Figure 5.14 Additional subprocess constraints (credit limit extension) for the full mortgage application process of Figure 5.2.

2If a subprocess adds new global events—as opposed to bound ones—these would appear with square corners inside the box.
The DCR Workbench: Declarative Choreographies for Collaborative Processes

Figure 5.15 Visualisation of additional subprocess constraints (credit limit extension) for the full mortgage application process of Figure 5.2; after one execution of Apply for limit extension.

Subprocess semantics is based on graph union; if $G = \ldots, e\{H\}$ contains a subprocess-spawning event $e$, then executing $e$ will form the graph $G \oplus H$, then apply the effects of $e$. Note the importance of bound names here: If events in $H$ were not bound, then repeated instantiation of the subprocess would not change the graph, i.e., $G \oplus H \oplus H = G \oplus H$. This equation emphatically does not hold under the current semantics, where events in $H$ may be bound, and thus replicated. In the running example, executing Apply for limit extension twice would result in all rounded-box event to be replicated twice—we invite the reader to try this out in the Workbench.

Adding subprocesses and bound events significantly increase the expressive power of DCR graphs [6, Theorem 9]:

**Theorem 1.** DCR graphs express the union of regular and $\omega$-regular languages. Graphs with subprocesses and bound events are Turing complete.
In particular, while event-reachability and refinement is decidable for plain and timed DCR graphs, they are undecidable for DCR graphs with subprocesses and bound events.

DCR graphs was extended with a notion of subprocesses and bound events in [12], followed by an investigation of expressive power in [6, 14]. The Workbench implements subprocesses in the sense of [12].

5.8 Data

The Workbench augments DCR graphs with a notion of “input events” and relations conditional on data. Suppose for our running example that if the amount applied for in a credit limit extension exceeds EUR 10,000, then having a bank statement becomes a condition for evaluating the loan application.

Technically, this is accomplished by: (a) adding the option of inputting a value when a subprocess is spawned; and (b) adding data-guards on select relations. When the subprocess is instantiated, condition on the input value dictates whether, e.g., the relation \( e \rightarrow f \) takes effect or not.

We extend the running example with such a conditional condition in Figure 5.16. Note that since the “variable” associated with an event is simply the name of the event, it becomes convenient to specify separately the name and label of the event. This is done in line 3, where it is specified that the event limit has label \( \text{Apply for a limit extension} \) and role Customer. (Without an explicit specification, the Workbench identifies event and label.)

The visualiser does not show data-guarded relations, and the formal semantics of DCR graphs with data have yet to be published.

```
limit["Apply for limit extension" role = Customer ] ?
{ /*Assess limit extension"*/
  /*Collect consent" -->> "Collect bank statement"
    "Submit budget"
    -->> !"Assess limit extension"
    -->>* "Assess loan application"
  "Collect bank statement"
  -->>* "Assess loan application"
  when "$limit > 10000"
}
-->> "Submit budget"
```

Figure 5.16 Alternative subprocess-with-data constraints (credit limit extension) for the full mortgage application process of Figure 5.2.
An interesting application of data is that of specifying user-input “forms” (think Web forms) via DCR graph, associating with each event in a graph an input field in such a form. This idea was implemented in the Actions panel and later realised [27] in collaboration with Exformatics A/S.

5.9 Other Panels

We mention here briefly a few panels of the Workbench not discussed so far.

1. An encoding from DCR to the GSM model [26] was defined in recent research [16]; the Workbench implements this encoding and outputs CMMN [4] XML.

2. Notions of concurrency and independence of DCR events, following the standard notion of labelled asynchronous transition systems [3, 36], was recently investigated [13]. The Workbench’ Concurrency panel implements these notions, automatically identifying concurrent events.

3. Work on applications of DCR in practical settings suggested a need for simplifying process models when presented to end users [9]. The Workbench contains a number of such simplifying views, most notably a “swimlane” view of the current trace in the panel of the same name, and a mechanism for projecting a graph in various ways to sub-graphs of interest.

5.10 Conclusion

We have given an introduction to DCR graphs, and an overview of the DCR Workbench. Since the Workbench implements most major variations of DCR graphs, this Chapter has also served as a survey of the state-of-the-art of DCR graphs as modelling and analysis tool for continuously changing distributed collaborative processes.

The Workbench has been instrumental for scientific research, providing a test-bed for quick experiments with new ideas; for teaching, providing students the opportunities for hands-on learning of abstract concepts; and for collaborations with industry and knowledge transfer to industry. In all these instances, providing a practical platform on which to demonstrate sometimes difficult-to-communicate abstract concepts helps to cement the reality and applicability of DCR as a modelling methodology. In particular, the Workbench has paved the way for academic results [9, 12, 18, 21, 27] to find their way to implementation in commercial tools [8, 28].
We invite the reader to use the Workbench for research and teaching. It is available at http://dcr.tools.

References


The DCR Workbench: Declarative Choreographies for Collaborative Processes


[34] Søren Debois and Tijs Slaats. The analysis of a real life declarative process. In CIDM 2015. IEEE, 2015. Accepted for publication.


