2014 IEEE 18th International Enterprise Distributed Object Computing Conference Workshops and Demonstrations

A Case for Declarative Process Modelling: Agile Development of a Grant Application System

Søren Debois, Thomas Hildebrandt and
Tijs Slaats
IT University of Copenhagen
Rued Langgaardsvej 7
2300 Copenhagen, Denmark
Email: {debois,hilde,tslaats}@itu.dk

Morten Marquard and
Tijs Slaats
Exformatics A/S
Copenhagen, Denmark
www.exformatics.com
Email: {mmq,ts}@exformatics.com

***Abstract*—We report on a recent industrial project carried out by Exformatics A/S in which the company used the declarative DCR Graphs notation to model and implement the grant application process of a Danish foundation. We present the process and discuss the advantages of the approach and challenges faced both while modelling and implementing the process. Finally, we discuss current work on extensions to the DCR Graphs notation aiming to address the challenges raised by the case study and to support the declarative, agile approach.**

I. Introduction

In the private as well as public sector, front-end customer services and their corresponding back-end processes and workflows are increasingly being digitalised by so-called Process Aware Information Systems (PAIS) [1]. The technology is pushed forward by promises of more usable, efficient and agile business processes, ensured to be compliant with business rules and the law.

Traditionally, functionality and user-experience of software systems have been described in a requirement specification *before implementation*. The requirements for functionality may be supported by semi-formal models such as e.g. UML sequence and activity diagrams and requirements for user-experience may be supported by use cases, user-interface descriptions and scenarios.

The new generation of PAIS are based on explicit, graphical and executable process models such BPMN [2], which supports a more agile development process, in which developers jointly with domain-experts *during implementation*, can design and simulate business processes and evaluate user-experiences.

However, the need for changes and adaptation to a PAIS does not end, when the system is being delivered to the customer. It is more the rule than the exception that the processes need to be adapted *after the system has been put into use*, either because the requirements where wrong or the needs changed before the final delivery. Indeed, often adaptations due to changes in business processes will be needed repeatedly throughout the entire lifetime of the system.

Moreover, it has been recognised in research, that imperative process descriptions such as BPMN that are based on explicit process flows tend to capture processes too rigidly, introducing un-justified dependencies and order between activities [1], [3], thereby increasing the need for adaptations to processes, even *during execution* of a process.

An explanation for this is, that the explicit flow graphs describe *how* a process is to be carried out, not *why*. By analogy to a way-finding service, it corresponds to describing a single route through a city, perhaps remembering to take a few possible exceptions into account. However, if the goal or map later changes, or the route is questioned regarding a potential improvement the flow-graph does not contain the information needed to derive a new route.

As an alternative, declarative process descriptions have been proposed by several research groups [3], [4], [5], [6], [7] as a way to achieve more flexible PAIS, by having process models that capture the *why* and not the *how*. The declarative approach has not seen wide-spread industrial adaptation yet and tool support has generally been limited to academic prototypes, which had led to the question if practitioners see actual opportunities to use declarative techniques. This has led to a study investigating what process modelling practitioners think of declarative modelling [8], which showed that the audience was receptive to the underlying concepts of the declarative modelling paradigm and the Declare [3], [4] and DCR Graphs notations, but also discovered that they had some difficulty getting used to the new paradigm and that the graphical notations used by the current notations were found to not always be intuitive, which made them relatively difficult to work with.

Exformatics A/S have during the past three years jointly with researchers at IT University of Copenhagen been developing, applying and implementing the declarative DCR graphs model [5], [6] in their standard Adaptive Case Mangement system. A key feature of DCR graphs is that they do not rely on a transformation before execution: The graphs describe both the initial state and the intermediate states, and thus support process adaptation during execution.

In the present paper we report on a recent industrial project in which Exformatics used the declarative DCR Graphs notation to model and implement the grant application process of a Danish foundation, the Dreyer foundation.

In Sec. II we first present the background for introducing a PAIS at the Dreyer foundation, and then in Sec. III we describe the key process(es) to be supported. In Sec. IV we first give the formal definition of DCR Graphs used in the Exformatics ECM Standard System, then we describe some of the sub processes of the grant application process in detail, showing how the models look like in the Exformatics process design tool, and finally we review the formal execution semantics of DCR Graphs. In Sec. V we briefly describe the Exformatics ECM system. We then discuss in Sec. VI the lessons learned during the implementation and discuss the advantages of the approach and challenges faced both while modelling and implementing the process. Finally, we conclude and discuss in VII current work on extensions to the DCR Graphs notation aiming to address the challenges raised by the case study and to support the declarative, agile approach.

II. The Grant Application Process

The Dreyer foundation handles applications from architects and lawyers in two annual application rounds. Previously, the foundation received the applications by mail and send them to an appropriate board member who is an expert in the domain of the application (that is, either architect or lawyer) for initial review. The number of applications is around 500-700 annually and the amount of paper being distributed is considerable. After the initial expert review, the applications are send to the other board members for review. This manual distribution and review process could easily take more than a month. After the reviews, a board meeting was held, typically lasting two long working days, to go through the applications and come up with a final recommendation. It was assumed that this process could be simplified by digitalising it. The aim is to be able to start voting earlier than today and enable the board to focus on the important applications at the board meeting, while simpler cases could be handled before the meeting as the members already had voted and agreed. The foundation had three major requirements for the solution:

1) Applications should be filled out by the applicants electronically in order to avoid lots of paper and facilitate easy distribution of the applications. The electronic form can be seen at http://formular.dreyersfond.dk/ (in Danish)
2) The board should be able to review and vote on the applications in an easy way.
3) The system should support the unique application case management process used by the foundation. The processes should be flexible and adaptive, in that it should be possible to handle exceptions within the system.

Exformatics responded to a request for proposal with its standard Adaptive Case Management tool which uses DCR graphs for supporting individual tailored adaptive processes. As mentioned in the second requirement above, it was a critical user experience requirement that board members could easily access, comment and vote on applications for a specific round. An idea of using traffic lights proposed by the foundation was prototyped, as shown in Fig. 1, demonstrating how the board members could easily view and change the status of each application.

III. The Process(es) to be Supported

Based on the task descriptions in the requirement specification and a few initial meetings with the foundation the following three processes were outlined as fundamental for supporting their case management process and requirements.

a) Round: Twice each year, the foundation has an application round. A round consists of a preparation, receiving, reviewing/voting, board meeting, payment, and a final closed *phase*. In each phase various events can happen. In preparation for a round, amounts are first reserved to be granted to architects and lawyers. When the round is initiated, applications are received. Applications have their own individual process life-cycle described below, including in particular the reviewing/voting done by four board members, whereof two acts as *experts*, one architect and one lawyer. After the application deadline, the decisions for all applications are finalised after a board meeting when the board meeting summary is approved. After this, applicants are informed, and each granted application enter the payment phase described in more detail below, until the last payment has been made.

b) Application: Each application is associated with a round and goes through a process of its own. After it is received an initial assessment is done by a case worker. If approved by the case worker, the application continues to an expert board member which does the first review, i.e. architect and lawyer applications are forwarded to the architect and lawyer board member respectively.

Once the expert board member has voted, the rest of the board can view the application and vote. The case worker can, provided the four board members agree, process the application before the board meeting. If so, the application is pre-approved or rejected. Other applications await the board meeting where all applications are voted for and decided by the board.

Once all applications are decided on, a summary of the board meeting is created and distributed among the board members. Once all approves the summary, the decision for all applications associated with the round is final, i.e. either approved or rejected, and the applicants are informed. If approved, a payment schedule and any need for further informations from the applicant are decided before the applicant is informed.

Once the supplementary information is received (if requested), a granted application continues to payment. By default, only a single payment is carried out. However, the case worker can decide to split the payment into several payments each with its own payment date. After delivery of the system, it was discovered, that if the applicants account number is modified during the application, the accountant must approve the modification before any payment can be done. As we will see, this could be handled by simple addition of a requirement to the process model and implementing a database trigger observing changes of account numbers.

Once all payments are completed, the application enters a feedback phase. In this phase, the applicant must send a summary of the project to the foundation. This information might be used in marketing material. Once the summary is received, the application is closed.

c) Payment: Each month the foundation collects the payments from each application that is ready to be paid out. The case worker review and approves the list, which is then transferred to Microsoft Dynamics (MD). The accountant is then asked to post the list and prepare and approve the payouts. After the accountant has done her work, the case worker is asked to approve the payouts. Once done the payouts are automatically transferred to the applicant through the integrated banking solution. Further, the case worker retrieves tax information about the applicants which is uploaded to the Danish tax authorities. Once all steps are done all payments are marked as completed. Any errors during the payout, e.g. invalid account numbers, are handled simply by manually removing the payment event from MD and modifying the account number.

IV. The DCR Graph Process Model

In this section, we first recall DCR graphs introduced in [5], [6] and further developed (e.g. by adding data) and implemented in the Exformatics ECM [9], [10], [11]. We then show how some of the sub processes in the Dreyer case management process described above where modelled as DCR Graphs.

Definition 1 (DCR Graph and DCR Graph with Data): A *DCR Graph* is a tuple $(\mathsf{E}, l, \mathsf{R}, \mathsf{M})$ where

- E is a finite set of *events* (the nodes of the graph).
- l is a *labelling* function, assigning a label to each event
- R is a finite set of relations between events (the edges of the graph). The relations are partioned into five kinds, the *conditions* ($\rightarrow\!\bullet$), *responses* ($\bullet\!\rightarrow$), *milestones* ($\rightarrow\!\diamond$), *inclusions* ($\rightarrow\!+$), and *exclusions* ($\rightarrow\!\%$).
- M is the *marking* of the graph. A marking represent the state of the process and is a triple $(\mathsf{Ex}, \mathsf{Re}, \mathsf{In})$ of sets of events, respectively the executed (Ex), the pending (Re), and the included (In) events.

A *DCR Graph with data* is a tuple $(\mathsf{E}, l, \mathsf{R}, \mathsf{M}_d, g, \mathsf{V})$ where V is a set of *data variables*, $\mathsf{M}_d = (\mathsf{M}, ev)$, $(\mathsf{E}, l, \mathsf{R}, \mathsf{M})$ is a DCR Graph, ev is an evaluation function mapping data variables to values, and g is a (partial) *relation guard* mapping from (some of) the relations to boolean expressions over variables.

Events are atomic and may represent the beginning or end of an activity, e.g. ”start application round” or ”review done”. An event may also represent a decision being made (e.g. ”vote reject”) or indeed an atomic event, e.g. account number changed. In the ECM implementation, data variables of DCR Graphs with data are mapped to entries in the case database, and events can be triggered from the case management user-interface, by the system itself or by triggers set up in the database. The database may be changed by events externally to the model. This provides a very flexible implementation, but also somewhat limits what can be verified formally.

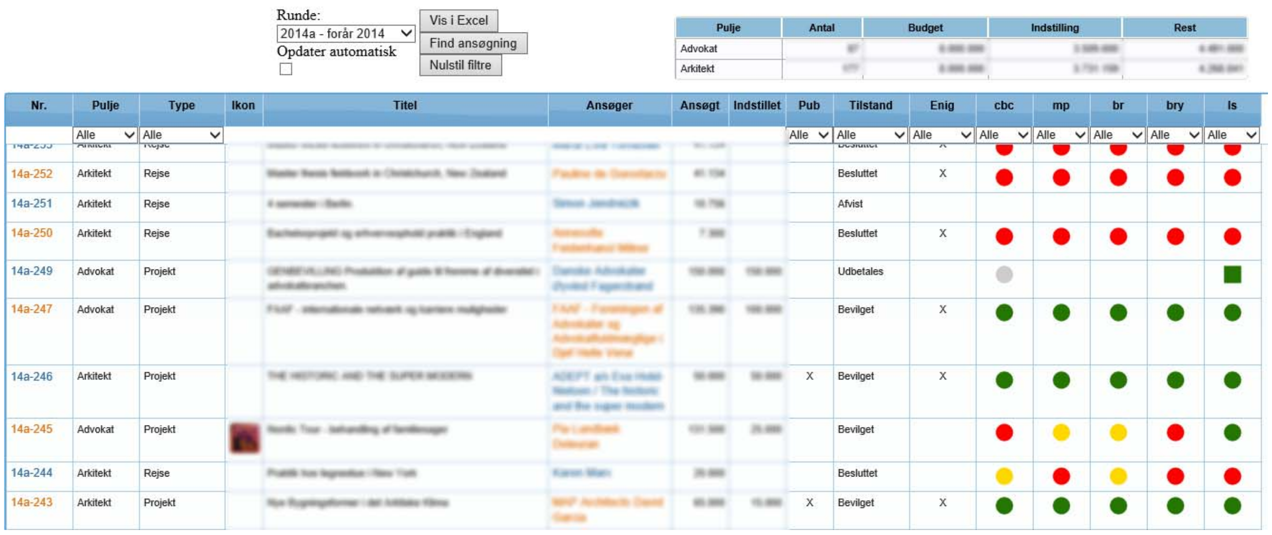
Before giving the formal semantics, we will give some example models of parts of the Dreyer process.

A. The review sub-process

As the name suggests, DCR graphs are *graphs*. They are represented visually in the DCRGraph editor [12] as in Fig. 2 illustrating the review sub process for an application.

Events (boxes) with labels (the text inside them) are related to other events by various arrows. If two events have the same label, the event ID is shown in italics in the bottom right corner. Otherwise, the editor chooses by default the event ID to be the same as the label and then it is not shown. In review process there are only four events, representing the four (submission of) review events by the four board members. The labels are Lawyer Review, Architect Review, Review, and Review. Note also that each box has an ”ear” with the role assignment.

Relations between the events govern their relative order of occurrence. When not constrained by any relations, events can happen in any order and any number of times. The review sub process only uses guarded condition relations. The guard expressions are referring to the variable R which indicates if



Runde: 2014a - forår 2014
Opdater automatisk ☐
Vis i Excel | Find ansøgning | Nulstil filtre

Pulje	Antal	Budget	Indstilling	Rest
Advokat	[illegible]	[illegible]	[illegible]	[illegible]
Arkitekt	[illegible]	[illegible]	[illegible]	[illegible]

Nr.	Pulje	Type	Ikon	Titel	Ansøger	Ansøgt	Indstillet	Pub	Tilstand	Enig	cbc	mp	br	bry	ls
	Alle	Alle						Alle	Alle	Alle	Alle	Alle	Alle	Alle	Alle
[illegible]	Arkitekt	Rejse		[illegible]	[illegible]	[illegible]			Besluttet	X	🔴	🔴	🔴	🔴	🔴
14a-252	Arkitekt	Rejse		[illegible]	[illegible]	[illegible]			Besluttet	X	🔴	🔴	🔴	🔴	🔴
14a-251	Arkitekt	Rejse		[illegible]	[illegible]	[illegible]			Afvist						
14a-250	Arkitekt	Rejse		[illegible]	[illegible]	[illegible]			Besluttet	X	🔴	🔴	🔴	🔴	🔴
14a-249	Advokat	Projekt		[illegible]	[illegible]	[illegible]	[illegible]		Udbetales		⚪				🟩
14a-247	Advokat	Projekt		[illegible]	[illegible]	[illegible]	[illegible]		Bevilget	X	🟢	🟢	🟢	🟢	🟢
14a-246	Arkitekt	Projekt		[illegible]	[illegible]	[illegible]	[illegible]	X	Bevilget	X	🟢	🟢	🟢	🟢	🟢
14a-245	Advokat	Projekt		[illegible]	[illegible]	[illegible]	[illegible]		Bevilget		🔴	🟡	🟡	🔴	🟢
14a-244	Arkitekt	Rejse		[illegible]	[illegible]	[illegible]			Besluttet		🟡	🔴	🟡	🔴	🔴
14a-243	Arkitekt	Projekt		[illegible]	[illegible]	[illegible]	[illegible]	X	Bevilget	X	🟢	🟢	🟢	🟢	🟢

Fig. 1. Board overview: Clicking on the round circles, grey, yellow, red or green, the board members can cast their vote and give a public and private comment.

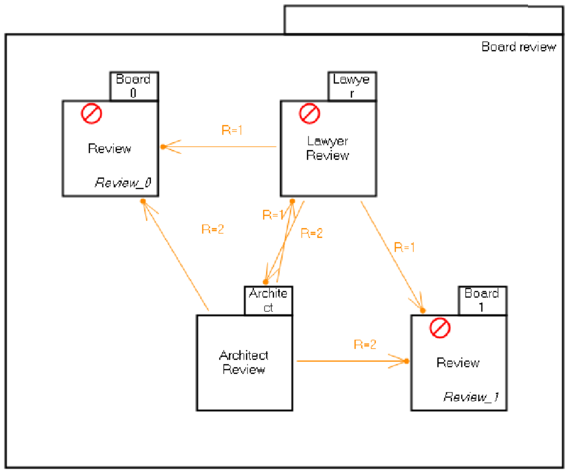


Fig. 2. A DCR graph for the review sub-process

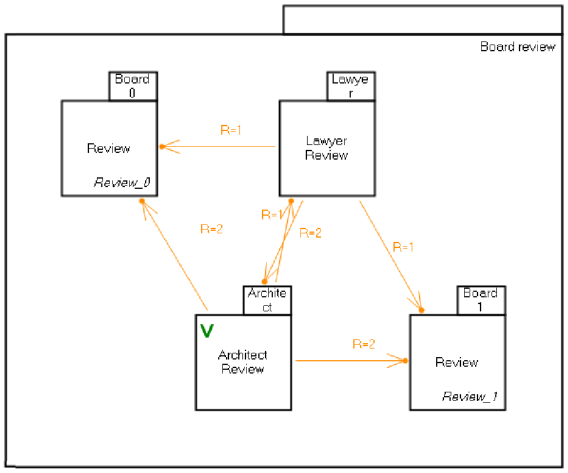


Fig. 3. A DCR graph for the review sub-process after the architect review

this is an application for Lawyer projects ($R = 1$) or Architect projects ($R = 2$). The meaning of the guarded relation is that it is only present if the guard is true. The figure shows the state if $R = 2$, and the "stop signs" indicate that Lawyer Review, Review, and Review are disabled because the guard on the condition from Lawyer Review, evaluates to true, while Architect Review is enabled because its constraint on the condition relation from Architect Review evaluates to false. The notation and semantics of the (unguarded) condition relation is similar to the precedence constraint in DECLARE [3], [4].

A key advantage of DCR graphs is that the graph *directly* represents the state of execution. There is no distinction between design-time and run-time. After executing the Architect review event, all events are possible. (Also Architect review, thereby allowing a new review to be submitted). The marking is visualised in the editor and simulation tool by adding a check mark to the Architect Review event indicating that it has been executed, as shown in Fig. 3.

B. The payment sub-process

A more complex example illustrating the use of all five kinds of relations is provided by the payment sub process in Fig. 7.

Changes of account numbers happen outside the case management system, but account numbers are registered in the database. By implementing a trigger that monitors account numbers in the database, any change can execute an event Account number changed in the case management process. Once this event is executed, the approve account number is made a required response by the (blue) response relation

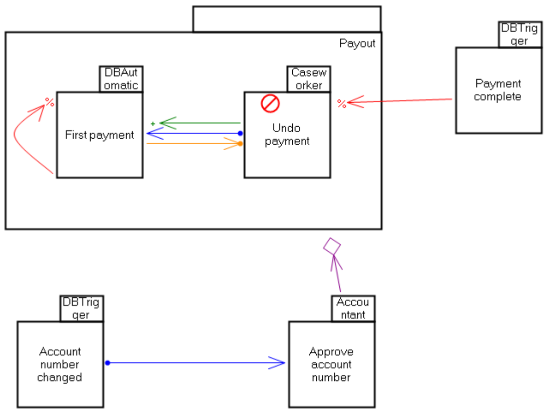


Fig. 4. A DCR graph for the payment sub-process

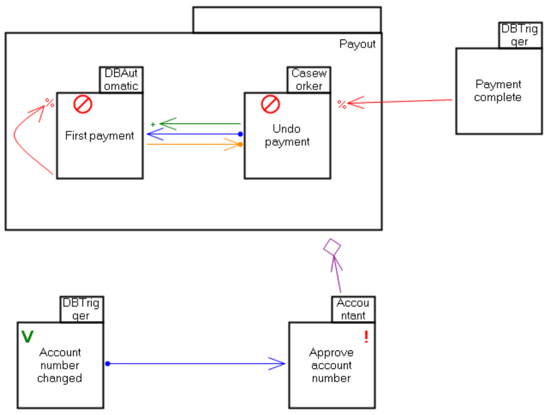


Fig. 5. A DCR graph for the payment sub-process after change of account number.

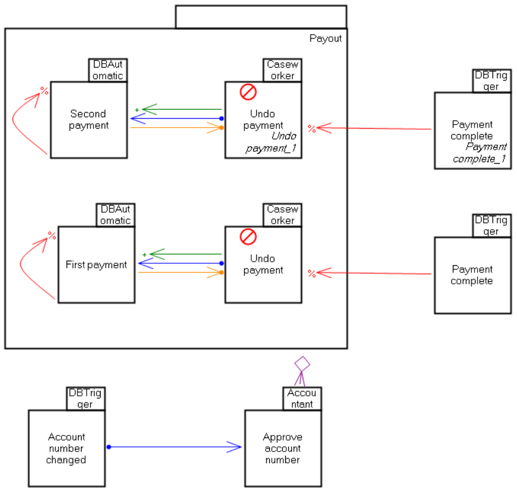


Fig. 6. A DCR graph for the payment sub-process with two payouts

with a bullet at the tail. In the marking, this is recorded by adding Approve Account Number to the set Re of pending responses. In the editor, it is visualised graphically by adding a (red) exclamation mark to the event, as shown in Fig. 5. As the approve account number event is a milestone, indicated by the (purple) relation with diamond at the head, for the payment phase, any payment event will be blocked until the approve account number has been executed, e.g. completed by the accountant.

The event First payment has role DBAutomatic, which means that it is carried out by the system and not a use (technically, it is scheduled for a date). Once it is executed it excludes itself, since it is related by the (red) exclusion relation with a $\%$ at the end to itself. This in turns enables a new event Undo payment allowing the case worker to undo a payment. If the Undo payment is executed, it includes the payment again, due to the (green) inclusion relation with a $+$ at the end, and makes it a required response, because of the (blue) response relation. Note that once the payment has been registered as completed (notified by the Payment complete database trigger, Undo payment is excluded and thus the payment can no longer be undone. The idea of this processes is, that if an error occurred in the payment (i.e. it is not completed) the error can be fixed by manually executing the Undo paymentevent, but once the payments is send to the bank without any errors the caseworker cannot redo the payment anymore by marking the payment as having an error.

C. Dynamic addition of sub processes

If the case worker adds one more payment to the application, the graph is dynamically modified resulting in the graph shown in Fig. IV-C

Notice that adding a payment result in three new events, a pattern of events, which has the same structure as the initial payment event. The pattern is added dynamically to the DCR graph modifying the logic. Notice also that Approve account number is a milestone for all the nested events in the Payout phase. Doing so simplifies the logic as any change to account number will block any payment event.

D. Cross-process synchronization

In the example of payment sub-processes above, dynamic addition of processes was handled by directly modifying the graph at runtime, and there were no relations connected directly to the payment sub-processes.

We can also add sub processes by starting new instances of processes, and synchronise execution through database triggers (i.e. events with role DBtrigger) and auto-execution events (i.e. events with role DBautomatic).

Such synchronization exists allowing applications to be pre-approved before handled formally at the board meeting, if the board cast 4 equal votes, or the board simply informs the case worker to pre-approve the application and start payment. At a certain point, such pre-approval cannot happen anymore, but must await board meeting summary approval, i.e. approval from all four board members that the meeting summary has been approved. The round therefore has an event Block Further

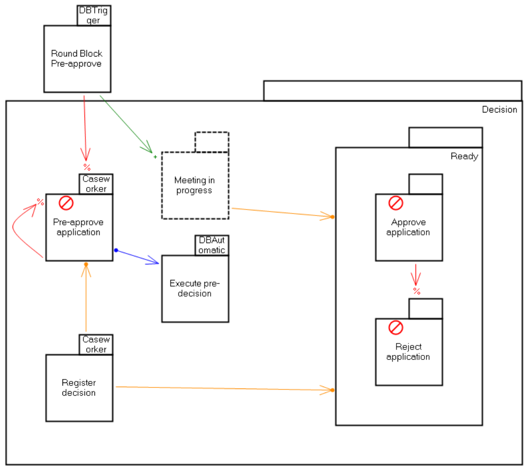
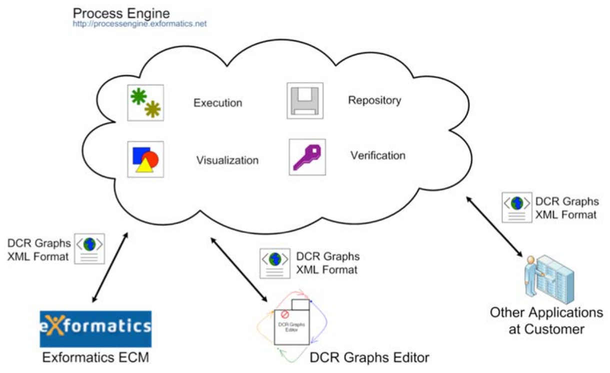


Fig. 7. A DCR graph for the payment sub-process



Decisions which once executed blocks the ability to pre-approve any applications. As the existing DCR model does not support cross-process events we have a database triggers which monitors this event. Once the Block Further Decision is executed the DBTrigger event Round block pre-approve is executed on all applications belonging to the round.

One this event is executed the event pre-approve application is excluded which means that the case worker cannot pre-approve the application anymore. Further the event Meeting in progress is included (green arrow). The reason for this event is to block the Ready nesting which contains the Approve and Reject application events. Using database triggers we can support events between processes. Each event acts as an interface for the process.

E. Formal Semantics

The formal execution semantics of DCR Graphs is given by notions of enabledness, transitions and acceptance. To keep the exposition simple we only give the formal semantics of plain DCR Graphs without data.

For a binary relation "$\rightarrow$" we write "$\rightarrow Z$" for the set $\{x \mid \exists z.\ x \rightarrow z\}$; similarly for "$Z \rightarrow$". We omit braces on singletons, writing $\rightarrow e$ rather than $\rightarrow\{e\}$.

Definition 2: Let $G = (\mathsf{E}, l, \mathsf{R}, \mathsf{M})$ be a DCR Graph, with marking $\mathsf{M} = (\mathsf{Ex}, \mathsf{Re}, \mathsf{In})$. We say that an event $e \in \mathsf{E}$ is *enabled* and write $e \in \mathsf{enabled}(G)$ iff (a) $e \in \mathsf{In}$ and (b) $\mathsf{In} \cap (\rightarrow\!\!\bullet e) \subseteq \mathsf{Ex}$, and (c) $\mathsf{In} \cap (\rightarrow\!\!\diamond e) \subseteq \mathsf{E} \backslash \mathsf{Re}$.

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

Definition 3: Let $G = (\mathsf{E}, l, \mathsf{R}, \mathsf{M})$ be a DCR Graph, with marking $\mathsf{M} = (\mathsf{Ex}, \mathsf{Re}, \mathsf{In})$. If $e \in \mathsf{enabled}(G)$ we may *execute* e obtaining the DCR Graph $H = (\mathsf{E}, \mathsf{R}, (\mathsf{Ex} \cup e, \mathsf{Re} \setminus e \cup (e\bullet\!\!\rightarrow), \mathsf{In} \setminus (e\rightarrow\!\%) \cup (e\rightarrow\!+))$. In this case we say that G has *transition on e to H* and write $G \xrightarrow{e} H$. A *run* of G is sequence $G = G_0 \xrightarrow{e_0} \cdots$. A run is *accepting* iff for all n with $e \in \mathsf{In}(G_n) \cap \mathsf{Re}(G_n)$ there exists $m \geq n$ s.t. either $e_m = e$, or $e \notin \mathsf{In}(G_m)$.

V. The Exformatics ECM Standard System

Exformatics Adaptive Case Management (ACM) enables knowledge workers to handle structured as well as unstructured information along a process. Both unstructured information, such as emails, documents and small messages in activity streams similar to Yammer and Facebook, and structured information, such as process responsible, application amount, account number, applicant name, is registered on the case, i.e. within the context of the process. A case can support different processes, each defined by a DCR graph. Exformatics ACM leverages Microsoft SharePoint and SQL Servers for hosting documents, emails and data. The front-end is build using REST services and jQuery based web parts hosted in a SharePoint portal. Applications are received using a semantics based form where the foundation can configure columns and rules, such as a mandatory column, themselves. Applications can contain documents which are stored in SharePoint document libraries. Processes are described using DCR graphs, and are executed by the Exformatics Process Engine, running as a cloud based web service. For each event executed, the DCR graphs of the process is extracted and send to the process engine for executing, and the resulting DCR graph is received and imported into the ACM platform. Any exchange of DCR graph use a common DCR xml standard briefly described in [13].

Such a model prepares processes for cross-corporate integration as only a few processes are intra-company only. Further DCR graphs are prepared to be used by other ACM systems.

The events can be exposed at the user-interface using customized visualization, as exemplified by the overview screen shown in FIg.1. The vote events for each application is linked directly to the trafic light bullets.

VI. Lessons Learned

In the following we describe some of the key lessons learned during the implementation of the adaptive case management system for the Dreyer foundation.

A. On-the-fly Process adaptation and creation of sub processes

Since the run-time-state is simply a marking of the graph, processes can be modified on-the-fly simply by changing the DCR graph. We benefited from this to allow for payment sub-processes to be dynamically added. Initially, support for one payment is included, which consist of the pattern of three events described above and shown in Fig. IV-C, which can deal with payment errors, e.g. due to errors in Microsoft Dynamics or the banking integration. End-users can, while the process is being executed, add more payments, which triggers adding a pattern of events as outlined above.

Since this is an expected adaptation, it could be have been good to have a way to model in the graph that sub-processes are dynamically added when an event, e.g. Add payment, is executed. Simliarly, each individual applications can be considered a sub-process of a round.

As the current DCR model does not support sub-processes, we had to implement synchronization between the round and the set of applications using database triggers. An example database trigger can be the Round Approved, which occurs on the application once the Round has been approved, which happens once the board meeting summary has been approved. Using a database trigger we fire this event for all applications of a given round once the round is approved. Creating such database triggers are not easy as we have to handle the concept of a sub-process outside the DCR logic. The ability to describe relations between a process and its subprocesses as formalized in the submitted paper [14] simplifies this logic, and the ability to describe a sub-process within the context of a process would ease the description and implementation.

A possible challenge when dealing with sub-processes identified in the present case study, is how to handle and formalize moving an application from one round to another. This sometimes happens if the board cannot decide what to do and postpone decision to the next board meeting. This would likely require formalizing passing identifies of events between processes, similar to the passing of names in the π-calculus.

This is not possible in the version of DCR Graphs implemented in the Exformatics tool. However, the theoretical extension of dynamic creation of sub processes called *Hierarchical DCR Graphs* has been described in a recently submitted paper [14] and implemented in a prototype, web-based simulation tool for DCR Graphs to be found at [15].

To handle *un-expected* additions to the process, we need support for end-users to define such patterns and add them to a running process, which in the simplest case would ammount to adding (or removing) a single event or relations to a graph. As described in [13], such run-time adaptations could in principle benefit from the formal verification supported by DCR Graphs, guaranteeing that adaptations are roboust, e.g. they do not introduce deadlocks, livelocks or other violations of specified properties. However, in practice this would require better verification techniques than what is currently implemented. Also, if extended with dynamic creation of sub processes, DCR Graphs becomes Turing-expressive [16] which in makes a general verification covering all possible processes impossible.

B. Iterative, agile process design

Defining and configuring the processes in DCR graphs was done in several phases where more and more details were added. During this process the process, e.g. the application, could be simulated in the DCR drawing tool, as well as in the system itself. This helped end-user understand the process and facilitated discussions on how the process should work and not work.

Despite a detailed analysis and many end user reviews, it (of course) turned out, that a requirement was still lacking in the designed and implemented process after delivery. The requirement that any modification to an account number requires accountant approval before payouts can be done was simply missing. Adding this requirement to the graph turned was very easy, as we just needed to adjust the DCR graph by adding the database trigger event event Account changed with a response relation to the approve account event, add logic to the customer object in the database to monitor changes to account numbers, and firing the Account changed if this occurs. Database triggers are simple to create and understand for such requirements and avoiding having the treatment of account numbers in the graph itself.

C. Understandability

DCR graphs as supported in the current editor are hard to read and edit. Support for compositional models and zooming in and out in details are needed for end users. Also, better descriptions of the DCR primitives and a modelling methodology is needed. The condition relation is easily understood, and the required response relation is also explainable. For the present case, milestones, inclusion and exclusions and also the nesting of events in phases were experienced as hard to explain and understand. Some other graphical representation for many-to-many relations than the nesting is likely needed. We also believe that it would be very useful if the system supported typical (happy) paths through the process, analalogous to a navigation system suggesting possible routes to a driver. The system should be able of adjusting the proposed paths to changing needs of the user, similar to how a navigation system can adapt to when a driver wants to make a detour to a gas station.

VII. Conclusion and Future Work

We have described an industrial project delivering an adaptive case management solution based on the Exformatics standard system and processes described as DCR Graphs.

DCR graphs have been a very valuable component to ensure Exformatics ACM can support the unique processes required by the foundation as well as run-time modifications to the processes as outlined in the payment event pattern. Rather than creating custom code the DCR graph describe the business rules and dependencies which enables us to support unique business requirements using a standard solution. As a product company this is important, as managing several code bases with unique customer requirements is too expensive.

The entire delivery was made in less than 4 months, from the first presentation of the requirement specification to the system as described in the present paper. The successful delivery demonstrated some of the claimed benefits of DCR Graphs in practice, in particular: 1) They support agile, rule based modelling that can be easily adapted when new requirements are discovered during and after delivery, 2) they support run-time addition of sub-processes which can be utilised for e.g. customised payment schemes, and 3) the resulting processes are allow maximal flexibility at run-time since they do not constrain the flow unnecessary - if some constraints are found unnecessary, they can simply be removed. The case study also demonstrated areas for future work and improvement, in particular: 1) support for dynamic creation of sub-processes should be part of the formal model, 2) if sub-processes are included in the model, formal verification of processes would me more likely to be possible, but we need to deal with the fact that the model becomes Turing-expressive, 3) the modelling notation, its graphical visualisation and design tools need to be improved to make the models more comprehensible and ultimately allow the business users to adapt processes themselves in a robust way. Enabling business users to take ownership of their processes, enables businesses to develop without changing the IT systems supporting the processes. As business processes evolves all the time we need to support such changes. Having to await a new product version from a vendor often does not suit the business requirement. Customer development does support the requirements either, as their require IT people for development and maintenance, is very time consuming and expensive.

Acknowledgment

This work was supported in part by the Computational Artifacts project (VELUX 33295, 2014-2017), by the Danish Agency for Science, Technology and Innovation, by Exformatics A/S, and by the IT University of Copenhagen. The authors would like to thank VisualIT for their contribution to the initial requirement elucidation.

References

[1] M. Reichert and B. Weber, *Enabling Flexibility in Process-Aware Information Systems: Challenges, Methods, Technologies*. Springer, 2012.

[2] Object Management Group BPMN Technical Committee, "Business Process Model and Notation, version 2.0," http://www.omg.org/spec/BPMN/2.0/PDF.

[3] W. M. van der Aalst and M. Pesic, "DecSerFlow: Towards a truly declarative service flow language," in *WS-FM 2006*, ser. LNCS, vol. 4184. Springer, 2006, pp. 1–23.

[4] W. van der Aalst, M. Pesic, H. Schonenberg, M. Westergaard, and F. M. Maggi, "Declare," Webpage, 2010, http://www.win.tue.nl/declare/.

[5] T. T. Hildebrandt and R. R. Mukkamala, "Declarative event-based workflow as distributed dynamic condition response graphs," in *PLACES*, ser. EPTCS, vol. 69, 2010, pp. 59–73.

[6] R. R. Mukkamala, "A formal model for declarative workflows: Dynamic condition response graphs," Ph.D. dissertation, IT University of Copenhagen, June 2012.

[7] R. Hull, E. Damaggio, F. Fournier, M. Gupta, F. T. Heath, S. Hobson, M. H. Linehan, S. Maradugu, A. Nigam, P. Sukaviriya, and R. Vaculín, "Introducing the guard-stage-milestone approach for specifying business entity lifecycles," in *WS-FM*, ser. LNCS, vol. 6551. Springer, 2010, pp. 1–24.

[8] H. A. Reijers, T. Slaats, and C. Stahl, "Declarative modeling-an academic dream or the future for bpm?" in *Proceedings of 11th International Conference on Business Process Management (BPM 2013)*, 2013, pp. 307–322.

[9] T. T. Hildebrandt, R. R. Mukkamala, and T. Slaats, "Safe distribution of declarative processes," in *SEFM*, ser. LNCS, vol. 7041. Springer, 2011, pp. 237–252.

[10] T. Slaats, R. R. Mukkamala, T. T. Hildebrandt, and M. Marquard, "Exformatics declarative case management workflows as DCR graphs," in *BPM*, ser. LNCS, vol. 8094. Springer, 2013, pp. 339–354.

[11] T. T. Hildebrandt, M. Marquard, R. R. Mukkamala, and T. Slaats, "Dynamic condition response graphs for trustworthy adaptive case management," in *OTM Workshops*, ser. LNCS, vol. 8186. Springer, 2013, pp. 166–171.

[12] T. Slaats, "DCR graph editor," 2013, http://www.itu.dk/research/models/wiki/index.php/DCR_Graphs_Editor.

[13] R. R. Mukkamala, T. Hildebrandt, and T. Slaats, "Towards trustworthy adaptive case management with dynamic condition response graphs," in *EDOC*. IEEE, 2013, pp. 127–136.

[14] S. Debois, T. Hildebrandt, and T. Slaats, "Hierarchical declarative modelling with refinement and sub-processes," *Submitted for publication*, 2014.

[15] S. Debois, "DCR exploration tool v.6," *IT University of Copenhagen*, 2014, http://www.itu.dk/research/models/wiki/index.php/DCR_Exploration_Tool.

[16] S. Debois, T. Hildebrandt, and T. Slaats, "Dynamic conditions, restless events and reproduction: ω-regular languages and beyond," *Submitted for publication*, 2014.