

Designing a Cross-organizational Case Management System using Nested Dynamic Condition Response Graphs

**Thomas Hildebrandt
Raghava Rao Mukkamala
Tijs Slaats**

**Copyright © 20xx, Thomas Hildebrandt
Raghava Rao Mukkamala
Tijs Slaats**

**IT University of Copenhagen
All rights reserved.**

**Reproduction of all or part of this work
is permitted for educational or research use
on condition that this copyright notice is
included in any copy.**

ISSN 1600–6100

ISBN 1111111111

Copies may be obtained by contacting:

**IT University of Copenhagen
Rued Langgaards Vej 7
DK-2300 Copenhagen S
Denmark**

Telephone: +45 72 18 50 00

Telefax: +45 72 18 50 01

Web www.itu.dk

Designing a Cross-organizational Case Management System using Nested Dynamic Condition Response Graphs

Thomas Hildebrandt
Raghava Rao Mukkamala
Tijs Slaats

1 Introduction

The purpose of a Case Management System as used in for instance Human Resource (HR) departments, hospitals, financial, and governmental institutions, is to guide case workers to perform the right tasks and to record the history of the case.

Since the initial work on office automation and workflow systems [9, 10, 32] it has been advocated to base the implementation of such systems, subsequently referred to as process-aware information systems [8], on explicit process descriptions described in some high-level process notation such as Petri Net or UML activity diagrams. The key motivations for using explicit process models are to allow the system to be more easily adapted to different work processes and to make the rules governing the system more visible to the users.

The rise of web service standards such as SOAP, WSDL and WS-BPEL has given new momentum to process-aware information systems. SOAP and WSDL standardize how to access external it systems as web services in a service oriented architecture and WS-BPEL provides a standard high-level programming language for combining individual service calls into process flows, also referred to as a process orchestration. Following WS-BPEL, the BPEL4People [1] and WS-HumanTask [20] specifications were the first attempt to standardize the inclusion of human tasks into BPEL to encompass workflows. Moreover, W3C started in 2004 developing the Web Services Choreography Description Language (WS-CDL) [29] which can be used to provide a global view of the intended interactions between different actors of a system, similar to the view of interactions provided by UML sequence diagrams. Within the last 5 years focus has moved from WS-BPEL and BPEL4People to the development of Business Process Model and Notation (BPMN) [19] which standardizes the graphical notation used for business processes, encompassing both human and automated tasks, and including both notations for orchestrations and choreographies.

However, as pointed out in e.g. [10, 27], the imperative process notations with explicit control and message flows underlying all of the above models describe the operationalization of business process goals and constraints, and not the goals and constraints themselves. Consequently, the notations are best suited for well-defined, rigid and repeatable workflows following a predefined sequence of service invocations and human tasks and one need to use ad hoc annotations to record the constraints and goals of the process. Moreover, it has proven to be non-trivial to support changes of the processes on-the-fly [26]. This does not match well the typical more ad-hoc nature of case work where it is often needed to redo and skip tasks and possibly adapt the set of tasks and their mutual constraints dynamically [25].

An alternative approach studied by several research groups is the use of declarative process models [3, 6, 22, 23, 27, 28], which describes the temporal constraints on process flows, not how to fulfill them. As part of the PhD project of the second author within the Trustworthy Pervasive Healthcare Services (TrustCare) research project [11] we have developed a declarative process model called Dynamic Condition Response Graphs (DCR Graphs) [12–14, 17]. The model is a both a generalization of the Process Matrix model [16, 18] developed by Resultmaker, a danish provider of workflow and case-management systems and the classical event structure model for concurrency [30,31]. DCR Graphs relates to DECLARE [28], which is a graphical notation that allow any temporal constraint pattern expressible as Linear-time Temporal Logic (LTL) formulas. However, instead of allowing the generality of expressing any constraint expressible in LTL, DCR Graphs only has a fixed handful of constraints which can be understood without reference

to LTL. Still it maintains the full expressive power of LTL (described in a follow up paper). Moreover, it is possible to give an operational semantics expressed directly as transitions between a novel type of markings of the tasks. In this way DCR Graphs combine the declarative view (constraints between tasks) with the imperative view (markings of tasks) allowing to trace the constraints even at run-time.

In the present paper we first briefly review the definition of DCR Graphs in Sec. 2 and then in Sec. 3 describe a case study of applying DCR graphs in the design phase of the development of a cross-organizational case management system. In Sec. 4 we briefly describe the current status of our development of tools for supporting design, simulation and verification of DCR Graphs. Finally, in Sec. 6 we outline challenges identified in the case study and the proposal for the continued development of the DCR Graphs model, technologies and tools further to make it applicable to component and model based design of distributed process-aware information systems.

2 Dynamic Condition Response Graphs

A Dynamic Condition Response Graph as introduced in [13] and extended in [14] consists of a set of events, a *marking* defining the execution state, and five binary relations between the events defining the conditions for the execution of events, the required responses and a novel notion of dynamic inclusion and exclusion of events. Hereto comes a set of actions, a labeling function assigning an action to each event, a set of roles, a set of principals and a relation assigning roles to actions and principals.

Formally we define a DCR Graph as follows.

Definition 1 A Dynamic Condition Response Graph is a tuple $(E, M, \rightarrow\bullet, \bullet\rightarrow, \rightarrow\diamond, \pm, \text{Act}, l, R, P, \text{as})$, where

- (i) E is the set of events
- (ii) $M \in \mathcal{M}(G) = \mathcal{P}_f(E) \times \mathcal{P}(E) \times \mathcal{P}(E)$ is the marking
- (iii) $\rightarrow\bullet \subseteq E \times E$ is the condition relation
- (iv) $\bullet\rightarrow \subseteq E \times E$ is the response relation
- (v) $\rightarrow\diamond \subseteq E \times E$ is the milestone relation
- (vi) $\pm : E \times E \rightarrow \{+, \%\}$ is a partial function defining the dynamic inclusion and exclusion relations by $e \rightarrow+ e'$ if $\pm(e, e') = +$ and $e \rightarrow\% e'$ if $\pm(e, e') = \%$
- (vii) Act is the set of actions
- (viii) $l : E \rightarrow \text{Act}$ is a labeling function mapping events to actions.
- (ix) R is a set of roles,
- (x) P is a set of principals (e.g. actors, persons, processors, services) and
- (xi) $\text{as} \subseteq (P \cup \text{Act}) \times R$ is the role assignment relation to principals and actions.

An event labelled with an action, e.g. **Create Case**, thus represents an execution of a (human or automated) task/activity/action **Create Case** in the workflow process. There may be several events with the same label, but in all our examples a label is assigned to a unique event, and thus we simply assume the set of events to be identical to the set of actions.

By default an event may be executed at any time and any number of times. However, the marking (defining the run-time state of the graph) and the five relations defined in (iii)-(vi) constrain the execution. The marking $M = (Ex, R, In) \in \mathcal{M}(G)$ consists of three sets of events, capturing respectively which events have *previously been executed* (Ex), which events are *pending responses required to be executed* (R), and finally which events are currently *included* (In). Only events $e \in In$, i.e. that are currently included, can be executed, and only if all currently included condition events e' , as specified by the condition relation $e' \rightarrow\bullet e$, have been executed and no currently included events e' which are milestones for e , as specified by the milestone relation $e' \rightarrow\diamond e$, are pending responses.

When an event e is executed it is added to the set of executed events (Ex) of the marking and all response events e' , as specified by the response relation $e \bullet \rightarrow e'$, are added to the set of responses R . Moreover, the set of included events is updated by adding (removing) all events e' included (excluded) by e as specified by the inclusion (exclusion) relation $e \rightarrow + e'$ ($e \rightarrow \% e'$).

The execution semantics of DCR Graphs is defined [12, 14] as a labelled transition system between markings as follows.

Definition 2 For a DCR Graph $G = (E, M, \rightarrow \bullet, \bullet \rightarrow, \rightarrow \diamond, \pm, l, \text{Act}, R, P, \text{as})$, we define the corresponding labelled transition systems $T(G)$ to be the tuple $(\mathcal{M}(G), M, \rightarrow \subseteq \mathcal{M}(G) \times \mathcal{L}(G) \times \mathcal{M}(G))$ where $\mathcal{L}(G) = E \times (P \times \text{Act} \times R)$ is the set of transition labels, $M = (Ex, In, R) \in \mathcal{M}(G)$ is the initial marking, $\rightarrow \subseteq \mathcal{M}(G) \times \mathcal{L}(G) \times \mathcal{M}(G)$ is the transition relation given by $M' \xrightarrow{(e,(p,a,r))} M''$

where

- (i) $M' = (Ex', In', R')$ is the marking before transition
- (ii) $M'' = (Ex' \cup \{e\}, In'', R'')$ is the marking after transition
- (iii) $e \in In'$, $l(e) = a, p \text{ as } r$, and $a \text{ as } r$,
- (iv) $\{e' \in In' \mid e' \rightarrow \bullet e\} \subseteq Ex'$,
- (v) $\{e' \in In' \mid e' \rightarrow \diamond e\} \cap R' = \emptyset$,
- (vi) $In'' = (In' \cup \{e' \mid e \rightarrow + e'\}) \setminus \{e' \mid e \rightarrow \% e'\}$,
- (vii) $R'' = (R' \setminus \{e\}) \cup \{e' \mid e \bullet \rightarrow e'\}$,

We define a run $(e_0, (p_0, a_0, r_0)), (e_1, (p_1, a_1, r_1)), \dots$ of the transition system to be a sequence of labels of a sequence of transitions $M_i \xrightarrow{(e_i, (p_i, a_i, r_i))} M_{i+1}$ starting from the initial marking. We define a run to be accepting if $\forall i \geq 0, e \in R_i, \exists j \geq i. (e = e_j \vee e \notin In_{j+1})$. In words, a run is accepting if no response event is pending forever, i.e. it must either happen at some later state or become excluded.

Condition (iii) in the above definition expresses that, only events e that are currently included, can be executed, and to give the label (p, a, r) the label of the event must be a , p must be assigned to the role r , which must be assigned to a . Condition (iv) requires that all condition events to e which are currently included should have been executed previously. Condition (v) states that the currently included events which are milestones to event e must not be in the set of pending responses (R'). Condition (vi) and (vii) are the updates to the sets of included events and pending responses respectively. Note that an event e' can not be both included and excluded by the same event e , but an event may trigger itself as a response.

In this paper we only consider finite runs. In this case, the acceptance condition degenerates to requiring that no pending response is included at the end of the run. This corresponds to defining all states where $R \cap In = \emptyset$ to be accepting states and define the accepting runs to be those ending in an accepting state. Infinite runs are also of interest especially in the context of reactive systems and the LTL logics. The execution semantics and acceptance condition for infinite runs are captured by mapping to a Büchi-automaton with τ -events and the work has been formalized in [12, 17].

2.1 Nested Distributed DCR Graphs with Milestones

In this section we first give the definition of nested distributed dynamic condition response graphs with milestone conditions. We then give the syntax for a textual process notation for nested distributed dynamic condition response graphs

Below we give the formal definition of the Nested DCR Graph model described informally above, which extends the model in our previous work [13] with nesting and the new milestone relation $\rightarrow \diamond$ between events. Note that the set of events is often the same as the set of actions and the labeling function is then the identity.

Definition 3 A Nested Distributed dynamic condition response graph with milestones is a tuple $(E, \triangleright, M, \rightarrow\bullet, \bullet\rightarrow, \rightarrow\diamond, \pm, \text{Act}, l, R, P, \text{as})$, where

- (i) E is the set of events
- (ii) $\triangleright : E \rightarrow E$ is a partial function mapping an event to its super-event (if defined), and we also write $e \triangleright e'$ if $e' = \triangleright^k(e)$ for $0 < k$, referred to as the nesting relation
- (iii) $M = (E, R, I) \subseteq \text{atoms}(E) \times \text{atoms}(E) \times \text{atoms}(E)$ is the marking, containing sets of currently executed events (E), currently pending responses (R), and currently included events (I).
- (iv) $\rightarrow\bullet \subseteq E \times E$ is the condition relation
- (v) $\bullet\rightarrow \subseteq E \times E$ is the response relation
- (vi) $\rightarrow\diamond \subseteq E \times E$ is the milestone relation
- (vii) $\pm : E \times E \rightarrow \{+, \%\}$ is a partial function defining the dynamic inclusion and exclusion relations by $e \rightarrow+ e'$ if $\pm(e, e') = +$ and $e \rightarrow\% e'$ if $\pm(e, e') = \%$
- (viii) Act is the set of actions
- (ix) $l : E \rightarrow \text{Act}$ is a labeling function mapping events to actions.
- (x) R is a set of roles,
- (xi) P is a set of principals (e.g. persons or processors) and
- (xii) $\text{as} \subseteq (P \cup \text{Act}) \times R$ is the role assignment relation to principals and actions.

where $\text{atoms}(E) = \{e \mid \forall e' \in E. e \triangleright e' \Rightarrow e' \neq e\}$ is the set of atomic events.

We require that the nesting relation $\triangleright \subseteq E \times E$ is acyclic and that there are no infinite sequence of events $e_1 \triangleright e_2 \triangleright \dots$. We will write $e \triangleright e'$ if $e \triangleright e'$ or $e = e'$, and $e \trianglelefteq e'$ if $e' \triangleright e$ or $e = e'$. We require that the nesting relation is consistent with respect to dynamic inclusion/exclusion in the following sense: If $e \triangleright e'$ or $e' \triangleright e$ then $\pm(e, e'') = +$ implies $\pm(e', e'') \neq \%$ and $\pm(e, e'') = \%$ implies $\pm(e', e'') \neq +$.

The new elements are the nesting relation $\triangleright \subseteq E \times E$ and the milestone relation $\rightarrow\diamond \subseteq E \times E$. The consistency between the nesting relation and the dynamic inclusion/exclusion is to ensure that when we map a nested DCR Graph to the corresponding flat DCR Graph as defined in Def. 4 below, no atomic event both includes and excludes another event. That is, if an event e includes (excludes) another event e'' , then any of its super or sub events e' can not exclude (include) the event e'' .

The new elements conservatively extend the DCR Graphs defined in [12] in the sense that given a Nested dynamic condition response graph as defined in Def. 3, the tuple $(\text{atoms}(E), M, \rightarrow\bullet, \bullet\rightarrow, \pm, \text{Act}, l, R, P, \text{as})$ is a (Distributed) dynamic condition response graph as defined in [12]. In particular, the semantics will be identical if both the \triangleright map and the milestone relation are empty.

We now define how to map a nested distributed dynamic condition response graph to a flat distributed dynamic condition response graph. Essentially, all relations are extended to sub events, and then only the atomic events are preserved. The labelling function is extended by labelling an atomic event e by the sequence of labels labelling the chain of super events starting by the event itself: $e \triangleright e_1 \dots e_k \not\triangleright$. The role assignment is extended to sequences of actions by taking the union of roles assigned to the actions.

Definition 4 For a Nested DCR Graph $G = (E, \triangleright, M, \rightarrow\bullet, \bullet\rightarrow, \rightarrow\diamond, \pm, \text{Act}, l, R, P, \text{as})$ define the underlying flat DCR Graph as

$$G^b = (\text{atoms}(E), M, \rightarrow\bullet^b, \bullet\rightarrow^b, \rightarrow\diamond^b, \pm^b, \text{Act}^+, l^b, R, P, \text{as}^b),$$

where $rel^b = \triangleright rel \trianglelefteq$ for some relation $rel \in \{\rightarrow\bullet, \bullet\rightarrow, \rightarrow\diamond\}$ and $\pm(e, e')^b = \pm(e_s, e'_s)$ if $\pm(e_s, e'_s)$ is defined and $e \triangleright e_s$ and $e' \trianglelefteq e'_s$ and $l^b(e_0) = a_0.a_1.a_2 \dots a_k$ if $e_0 \triangleright e_1 \triangleright e_2 \dots \triangleright e_k$ and $l(e_i) = a_i$ for $0 \leq i \leq k$ and $\text{as}^b(a_0.a_1.a_2 \dots a_k) = \{\text{as}(a_i) \mid 0 \leq i \leq k\}$ and $\text{as}^b(p) = p$ for $p \in P$.

It is easy to see from the definition that the underlying DCR Graph has at most as many events as the nested graph and that the size of the relations may increase by an order of n^2 where n is the number of atomic events.

3 Case Study: A Cross-Organizational Case Management System

In this section we demonstrate how we have applied DCR Graphs in practice within a project that our industrial partner Exformatics carried out for one of their customers. In the process, we acted as consultants, applying DCR Graphs in meetings with Exformatics and the customer to capture the requirements in a declarative way, accompanying the usual UML sequence diagrams and prototype mock-ups. Sequence diagrams typically only describe *examples* of runs, and even if they are extended with loops and conditional flows they do not capture the constraints explicitly.

The customer of the system is *Landsorganisationen i Danmark* (LO), which is the overarching organization for most of the trade unions in Denmark. Their counterpart is *Dansk Arbejdsgiverforening* (DA), which is an overarching organization for most of the danish employers organizations.

At the top level, the workflow to be supported is that a case worker at the trade union must be able to create a case, e.g. triggered by a complaint by a member of the trade union against her employee. This must be followed up by a meeting arranged by LO and subsequently held between case workers at the trade union, LO and DA. After being created, the case can at any time be managed, e.g. adding or retrieving documents, by case workers at any of the organizations.

Fig. 1 shows the graphical representation of a simple DCR Graph capturing these top level requirements of our case study.

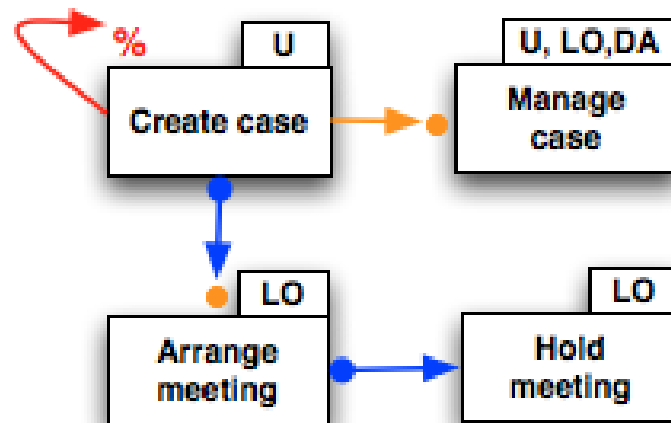


Figure 1: Top level requirements as a DCR Graph

Four top-level events were identified, shown as boxes in the graph labelled Create case, Manage case, Arrange meeting and Hold meeting. For the top-level events we identified the following requirements:

1. A case is created by a union case worker, and only once.
2. The case can be managed at the union, LO and DA after it has been created.
3. After a case is created, LO can and must arrange a meeting between the union case worker, the LO case worker and the DA case worker.
4. After a meeting is arranged it must be held (organized by LO).

The requirements translate to the following DCR Graph role assignments (shown as "ears" on the event boxes) and relations shown as different types of arrows between the events in Fig. 1:

1. Create case has assigned role U and excludes itself.

2. Create case is a condition for Manage case, which has assigned role U, LO and DA.
3. Create case has Arrange meeting as response, which has assigned role LO.
4. Arrange meeting has Create case as a condition and Hold meeting as response, which has assigned role LO.

For example, the U on Create case indicates that only a case worker at the trade union (U) can create a case, and the U, LO, DA on Manage case indicate that both the trade union, LO and DA can manage the case.

The arrow Create case → • Manage case denotes that Manage case has Create case as a (pre) *condition*. This simply means that Create case must have happened before Manage case can happen. Dually, Arrange meeting has Hold meeting as *response*, denoted by the arrow Arrange meeting • → Hold meeting. This means that Hold meeting *must* eventually happen after Arrange meeting happens. Finally, the arrow Create case → % Create case denotes that the event Create case excludes itself.

In the subsequent meetings, we came to the following additional requirements:

1. (a) To create a case, the case worker should enter meta-data on the case, inform about when he/she is available for participating in a meeting and then submit the case.
 (b) When a case is submitted it may get a local id at the union, but it should also subsequently be assigned a case id in LO.
 (c) When a case is submitted, LO should eventually propose dates.
2. (a) Only after LO has assigned its case id it is possible to manage the case and for LO to propose dates.
 (b) Manage case consists of three possible activities (in any order): editing case meta data, upload documents and download documents. All activities can be performed by the Union, LO and DA.
3. (a) The meeting should be arranged in agreement between LO and DA: LO should always propose dates first - and then DA should accept, but can also propose new dates. If DA proposes new dates LO should accept, but can also again propose new dates. This could in principle go on forever.
 (b) The union can always update information about when they are available.
4. (a) No meeting can be held unless there is an agreement on the meeting dates.

These requirements led to the extension of the model allowing *nested* events as recalled in the previous section and given in full detail in [14].

The requirements could then be described by first adding the following additional events to the graph: A new super event Edit which has the subevents: Metadata and Dates available. The Create case event has two sub events: Submit case and Assign case Id. The Manage case event has three sub events: Edit metadata, Upload and Download. The Arrange meeting event has four sub events: Propose dates-LO, Propose dates-DA, Accept LO and Accept DA.

Subsequently, the relations was adapted to the following (Nested) DCR Graph relations, as shown in Fig 2:

1. Edit is a condition to Submit case and is assigned role U.
2. Within the Create case superevent:
 - (a) Submit case is a condition to Assign case Id and also requires it as a response.
 - (b) Assign case Id is a condition for Manage case (and therefore also all it's sub events).
 - (c) Assign case Id is now the condition for Propose dates-LO and Submit case requires it as a response.
3. Within the Arrange meeting superevent:
 - (a) Arrange meeting still has Hold meeting as response, but is now also required as a milestone for Hold meeting
 - (b) Propose dates-LO is a condition for Propose dates-DA

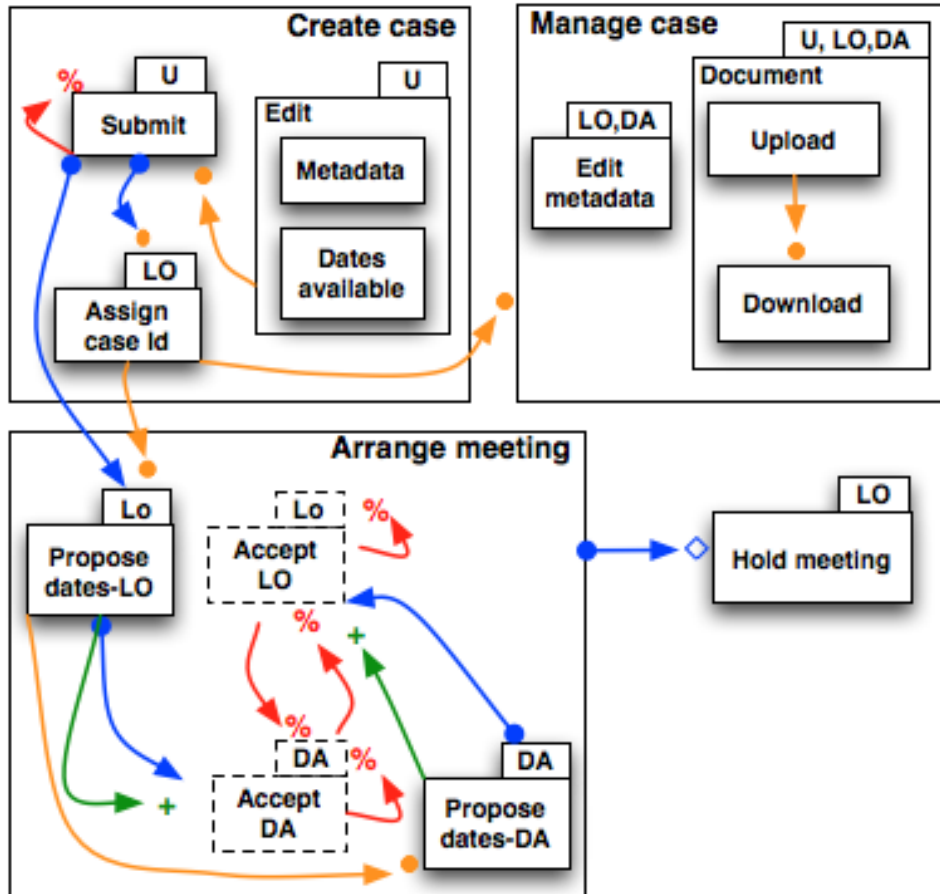


Figure 2: Case Handling Process

- (c) Propose dates-LO includes Accept DA and requires it as a response
 - (d) Propose dates-DA includes Accept LO and requires it as a response
 - (e) Accept LO excludes itself and Accept DA
 - (f) Accept DA excludes itself and Accept LO
4. Within the Manage case superevent:
- (a) Edit metadata has roles LO and DA assigned to it.
 - (b) Upload and Download have been grouped under a superevent Document with roles U, LO and DA assigned to it.
 - (c) Upload is a condition for Download.

During the case study it became clear that it would be useful to have design tools allowing to quickly create and simulate models. In the following section we describe the tools developed so far. In Sec. 6 we describe the plans for future development of tools along with the challenges for extending the theory identified in the case study.

4 Prototype tools

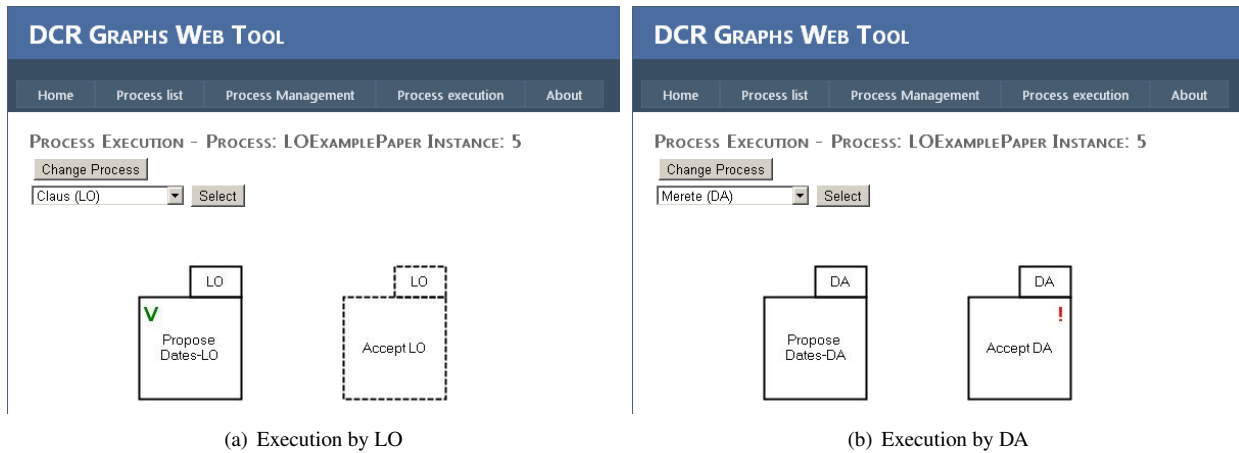


Figure 3: Execution in the Web Tool

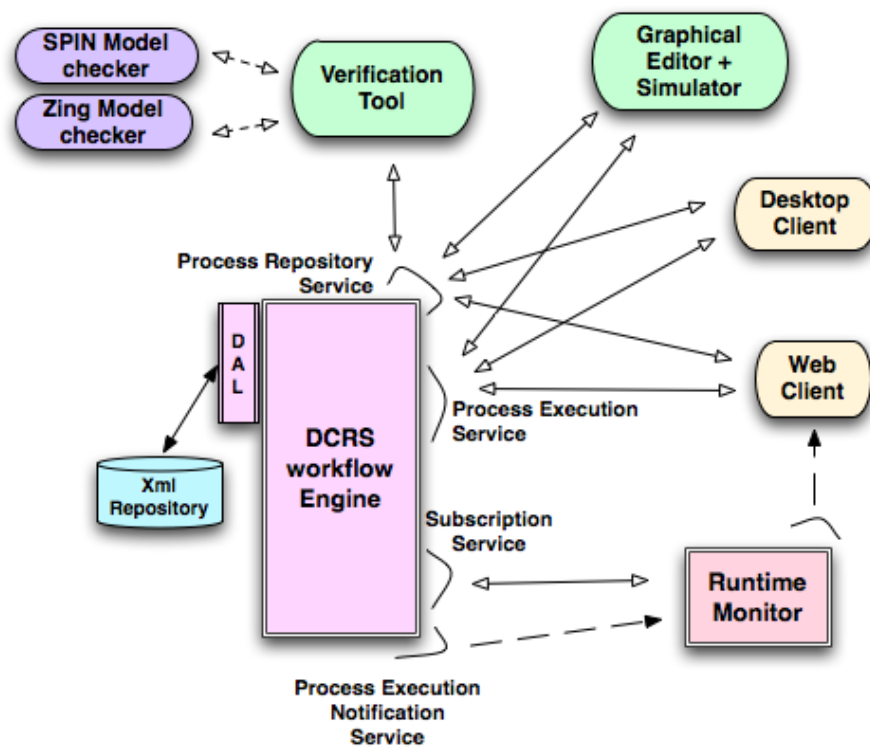


Figure 4: Prototype Architecture

To support designing with DCR Graphs, making the model available to a wider audience and allow interested parties to experiment with the notation, we are developing prototype implementations of various tools for DCR Graphs. Development up to this point includes:

1. A process repository; a service which can be used to store and retrieve DCR processes and process instances.
2. An execution host; a service which can be used to execute DCR process instances.
3. A windows-based graphical editor; which can be used to model DCR Graphs and run simple simulations on them.
4. A windows-based desktop client for executing process instances.
5. A platform independent web client; which can also be used to execute process instances. In the future we aim to support the creation of processes through this webinterface as well. (Fig. 3)
6. A model checking and runtime verification tool; which interfaces to SPIN [24] and ZING [21] model checkers for model checking.
7. A runtime-monitor that can subscribe to the execution host and verify that the execution of processes adheres to given properties.

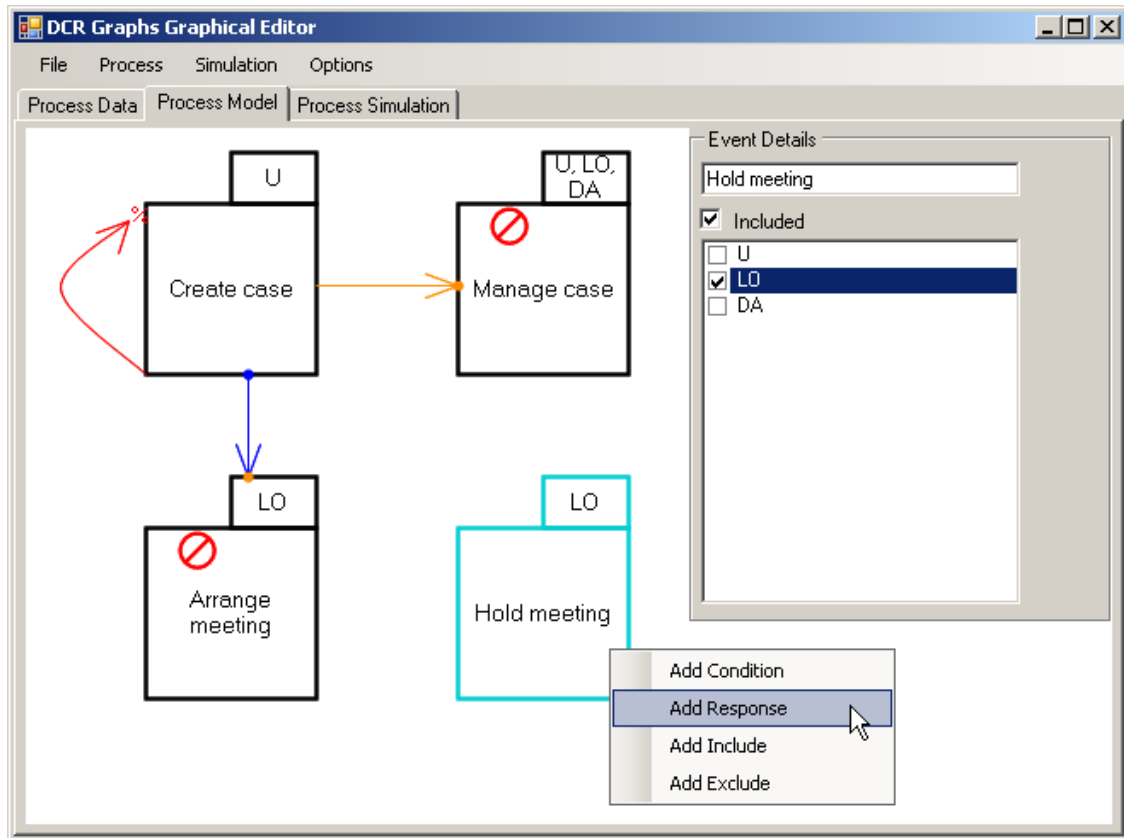


Figure 5: The Graphical Editor

Fig. 4 shows how these tools interact: Usually, a process modeller will first create a process in the graphical editor (Fig. 5), which will be stored in the process repository. The process modeller can use the verification tool to check if his process adheres to the properties that he desires. Both safety and liveness properties on models can be verified with the help of SPIN [24] model checker, where as only safety properties on DCR Graphs can be verified using ZING [21] model checker, as ZING does not support liveness properties. A user can login to the web or desktop-client and select

the process for execution. The client will request that the process repository start a new instance and the repository will provide the client with the description of the process and runtime information on the process instance. Execution requests are made to the execution server, which handles these requests atomically, making updates to the instance stored on the repository. If a request is invalid, the execution server will notify the user and leave the process instance in its original state. The runtime monitor can subscribe to the execution server and will get notified of every execution request. It will then check if the execution of the process follows the properties described for it.

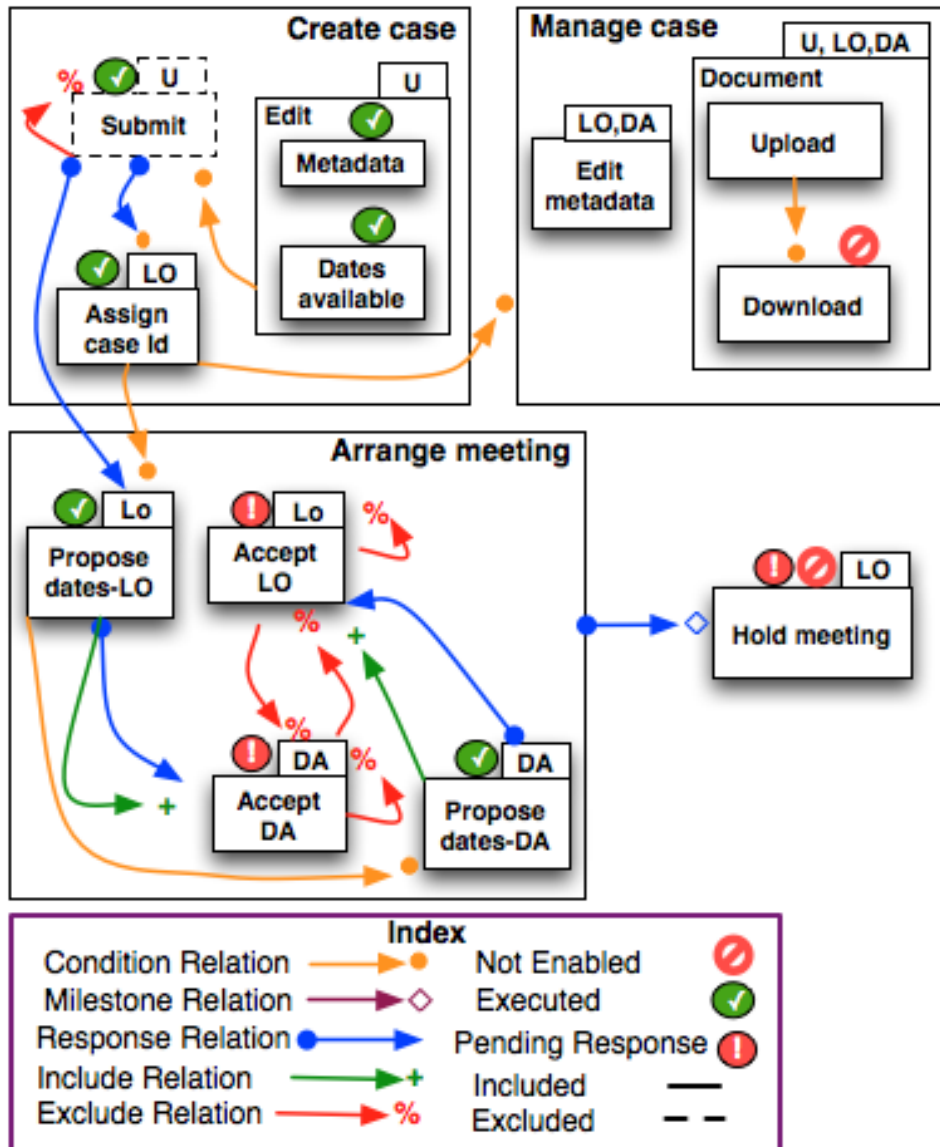


Figure 6: Case Handling Process Runtime

All the prototype tools support the basic DCR Graph notation containing condition, response, include and exclude relations. We are currently working on extending the prototype to support milestone relations and nested events. In Fig. 6, 7, 8, we have illustrated how the execution state of the case-handling process may be visualized in the simulator in the future.

The graph in the figure. 6 shows the state after a run where the union started by creating a case: they edited meta-data, indicated the dates they were available and submitted. When LO received the case they assigned their own case ID to it. Some time later LO proposed possible dates for a meeting to DA. DA did not agree with these dates and responded by proposing some of their own. In the graph both Accept LO and Accept DA are included and have a pending response because both LO and DA have proposed dates. Because of these pending responses Hold meeting is disabled. Because no files have been uploaded to the document yet, Download is also disabled.

The graph in the figure. 7 shows the runtime state after the union has uploaded an agenda for the meetings. Note that, since the union has uploaded a file to the case, the Download is now enabled. But at the same time, Accept LO and Accept DA still remains the same as the previous graph, as the proposed dates have not been accepted yet by either LO or DA.

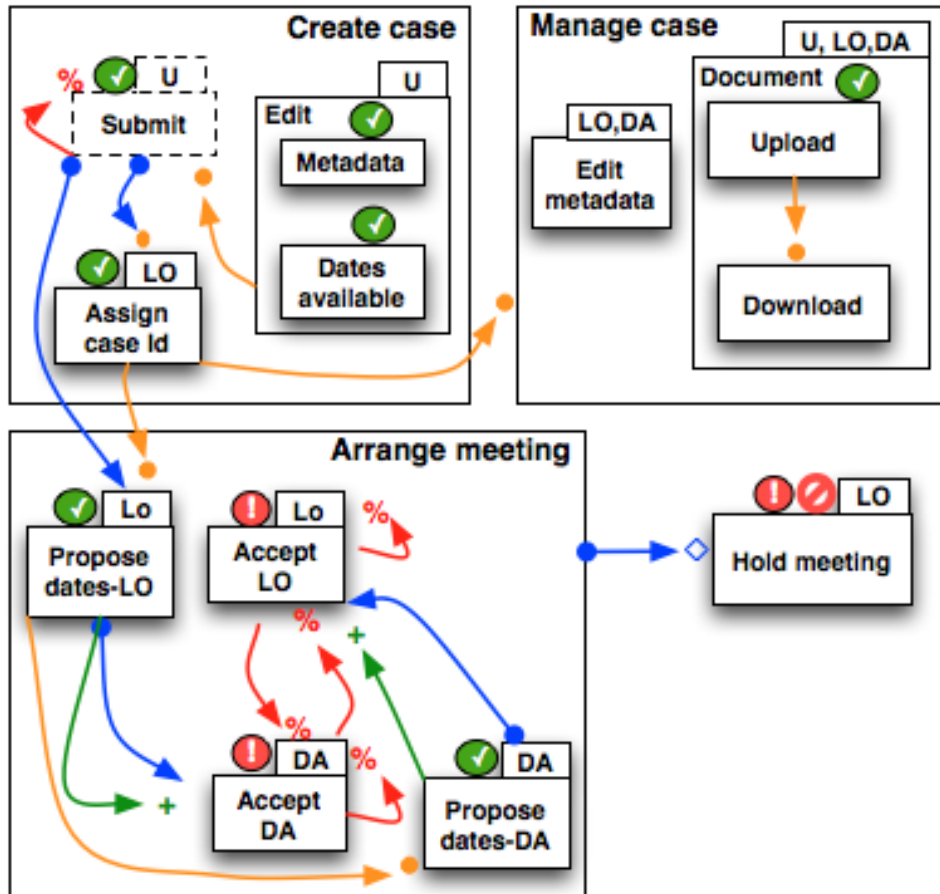


Figure 7: Case Handling Process Runtime After Upload Document

Figure 8 shows the graph representing the state after LO has accepted one of the dates proposed by DA. Note that both Accept LO and Accept DA are excluded due to a mutual exclude relation between them. Even though there is a pending response on Accept DA, it is not considered relevant as it is excluded and Hold meeting has become enabled with a pending response. At this state Hold meeting by LO will make the graph to reach the accepting state, as there will be no included pending responses.

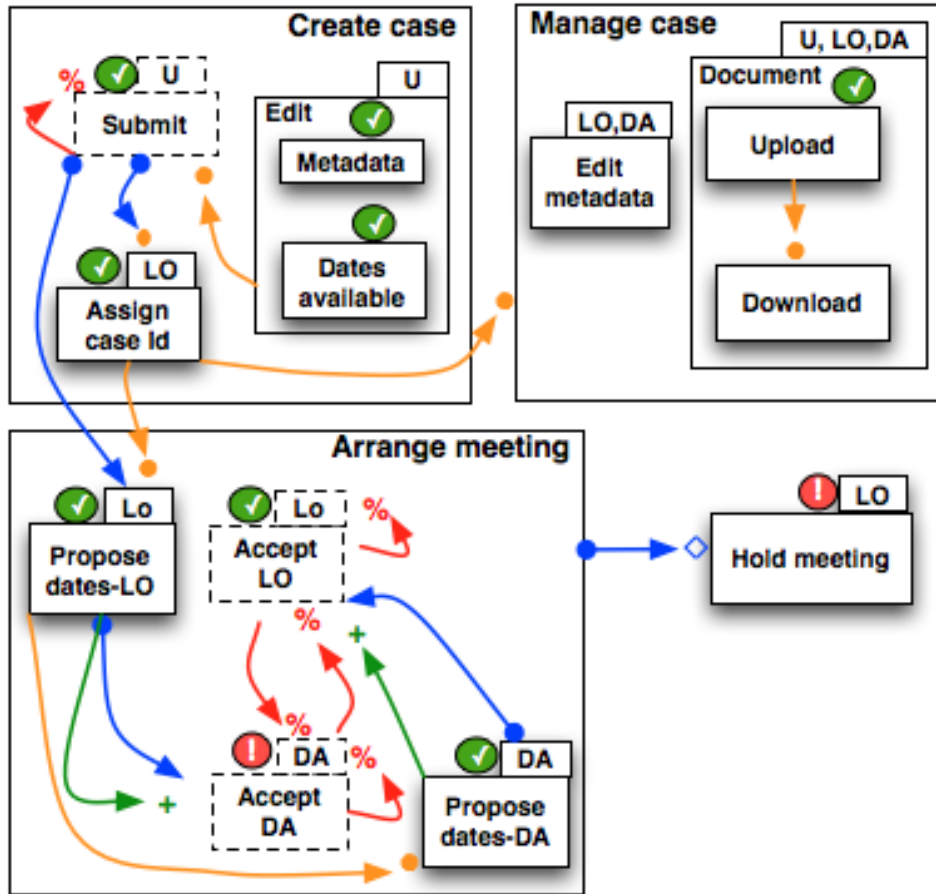


Figure 8: Case Handling Process Runtime After Accept Dates

5 Comparison to other Approaches

As already mentioned in the introduction, our approach is closely related to the work on DECLARE [27, 28]. In particular the condition and response relations are also considered in [27, 28], and we have used the same graphical notation as loc. cit. The crucial difference is that we focus on a few core constraints allowing to describe the state and operational semantics of processes as a labelled transitions between simple markings consisting of three sets of respectively executed, included and required events. As also pointed out in [27, 28], the generality of LTL offers much flexibility with respect to specifying execution constraints but makes it more complex to execute processes given in DECLARE and to describe and understand their run-time state. It typically requires a translation of the constraints to LTL and subsequent using the standard mapping of an LTL formula to a Büchi-automaton. In particular, there is no obvious way to trace the graphical constraints in DECLARE to the states of the Büchi-automaton. Moreover, we show in a follow up paper that every set of traces expressible in LTL (and thus DECLARE) can also be expressed using DCR Graphs.

We have shown in [18] it is possible, but much more complex to represent in LTL the interplay between dynamic inclusion/exclusion and the other relations. Neither this novel notion of dynamic inclusion/exclusion relations nor nesting are considered in [27, 28].

The DCR Graphs model also relates to the independent work on the *Guard-Stage-Milestone* model [15] by Hull et al presented as an invited talk at the WS-FM 2010 workshop and part of the work on artifact-centric business

process [2, 5, 7].

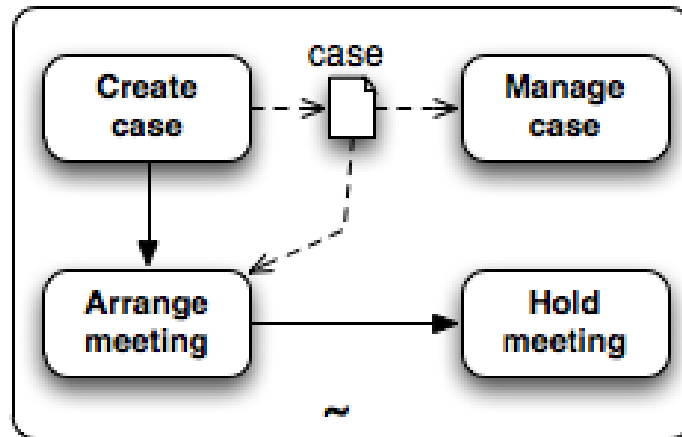


Figure 9: BPMN 2.0 ad-hoc sub-process activity

Finally, BPMN 2.0 includes the *ad-hoc sub-process activity* which allows one to group a set of activities that can be carried out in an ad-hoc way. Fig. 9 below we show how one may attempt to describe the top level requirements described in Fig. 1 as a BPMN 2.0 ad-hoc sub-activity. According to the informal description of BPMN 2.0 ad-hoc sub-process activity in the current BPMN 2.0 specification ([19]) *Create case* is a *condition* for *Manage case* since the latter cannot start without the *data object case* as input, which is produced by the former. Moreover, quoting from the specification, the sequence flow between *Create case* and *Arrange meeting* (and similarly between *Arrange meeting* and *Hold meeting*): “creates a dependency where the performance of the first Task *MUST* be followed by a performance of the second Task. This does not mean that the second Task is to be performed immediately, but there *MUST* be a performance of the second task after the performance of the first Task.”. This seems exactly to correspond to the response relation in DCR Graphs. However, when reading the semantics section of the specification ([19], Sec.13.2.5, 445-446) it appears that the sequence flow introduces just a standard precondition. Thus, the specification is not consistent in the description of sequence flows within ad-hoc sub-activities. Also, it is not clear how to specify roles on actions (swim lanes seem not to be allowed within ad-hoc sub-activities) nor how to specify that an activity within an ad-hoc sub activity only can be executed once. In particular, *Create case* can be executed any number of times in the above process.

6 Conclusions and Future Work

Our case study showed that DCR Graphs model is well suited to give a global description of the temporal constraints between the individual tasks which is helpful in capturing the requirements of the overall system.

However, there are still many points for future developments.

First of all there is the need to extend the expressiveness of DCR Graphs. In the ongoing PhD project of the second author we intend to extend the DCR Graphs model to be able to express relevant features such as multi-instance sub-graphs (allowing the dynamic creation of sub-graphs representing dynamic sub process instantiation), time, exceptions and data. Along with this we intend to continue developing the technology for model checking and run time verification and apply it within case studies.

Second, our industrial partner Resultmaker who already use a declarative process model based on the primitives in the DCR Graphs model expects to investigate the use of the formalization to support *safe* dynamic changes to the process constraints at run time.

Thirdly, the DCR Graphs model presently describe a global view of the process. Through our discussions with Exformatics during the case study we identified the wish to be able to automatically synthesize distributed views of the process. In particular, they wanted to be able to derive descriptions of communication protocols and message exchange between the individual local components in a distributed implementation of the system.

Derivations of descriptions of communication protocols between local components from a global model is been researched for the imperative choreography language WS-CDL in the work on structured communication-centred programming for web services by Carbone, Honda and Yoshida [4]. Put briefly, the work formalizes the core of WS-CDL as the global process calculus and define a formal theory of end point projections projecting the global process calculus to abstract descriptions of the behavior of each of the local "end-points" given as pi-calculus processes typed with session types.

We are currently working on the challenge of synthesizing a distributed view of a DCR Graphs as a set of interacting DCR Graphs, thus providing a declarative notion of end-point projections. As a challenge for future work we propose to provide a formal map between DCR Graphs and imperative choreographies formalized in the global process calculus [4].

References

- [1] Active Endpoints, Adobe Systems, BEA Systems, IBM, Oracle, SAP. Ws-bpel extension for people (bpel4people) version 1.0, 2007. http://www.adobe.us/content/dam/Adobe/en/devnet/livecycle/pdfs/bpel4people_spec.pdf.
- [2] Kamal Bhattacharya, Cagdas Gerede, Richard Hull, Rong Liu, and Jianwen Su. Towards formal analysis of artifact-centric business process models. In *In preparation*, pages 288–304, 2007.
- [3] Christoph Bussler and Stefan Jablonski. Implementing agent coordination for workflow management systems using active database systems. In *Research Issues in Data Engineering, 1994. Active Database Systems. Proceedings Fourth International Workshop on*, pages 53–59, Feb 1994.
- [4] Marco Carbone, Kohei Honda, and Nobuko Yoshida. Structured Communication-Centred Programming for Web Services. In *16th European Symposium on Programming (ESOP'07)*, LNCS, pages 2–17. Springer, 2007.
- [5] David Cohn and Richard Hull. Business artifacts: A data-centric approach to modeling business operations and processes. *IEEE Data Eng. Bull.*, 32(3):3–9, 2009.
- [6] Hasam Davulcu, Michael Kifer, C. R. Ramakrishnan, and I.V. Ramakrishnan. Logic based modeling and analysis of workflows. In *Proceedings of ACM SIGACT-SIGMOD-SIGART*, pages 1–3. ACM Press, 1998.
- [7] Alin Deutsch, Richard Hull, Fabio Patrizi, and Victor Vianu. Automatic verification of data-centric business processes. In *Proceedings of the 12th International Conference on Database Theory, ICDT '09*, pages 252–267, New York, NY, USA, 2009. ACM.
- [8] Marlon Dumas, Wil M. van der Aalst, and Arthur H. ter Hofstede. *Process Aware Information Systems: Bridging People and Software Through Process Technology*. Wiley-Interscience, 2005.
- [9] Clarence A. Ellis and Gary J. Nutt. Office information systems and computer science. *ACM Comput. Surv.*, 12:27–60, March 1980.
- [10] Clarence A. Ellis and Gary J. Nutt. Workflow: The Process Spectrum. In Amit Sheth, editor, *Proceedings of the NSF Workshop on Workflow and Process Automation in Information Systems*, pages 140–145, May 1996.
- [11] Thomas Hildebrandt. Trustworthy pervasive healthcare processes (TrustCare) research project. Webpage, 2008. <http://www.trustcare.dk/>.
- [12] Thomas Hildebrandt and Raghava Rao Mukkamala. Declarative event-based workflow as distributed dynamic condition response graphs (submitted for journal publication), 2010.

- [13] Thomas Hildebrandt and Raghava Rao Mukkamala. Distributed dynamic condition response structures. In *Pre-proceedings of International Workshop on Programming Language Approaches to Concurrency and Communication-centric Software (PLACES 10)*, March 2010.
- [14] Thomas Hildebrandt, Raghava Rao Mukkamala, and Tijs Slaats. Nested dynamic condition response graphs. In *Fundamentals of Software Engineering Conference 2011 (to appear)*, April 2011.
- [15] Richard Hull. Formal study of business entities with lifecycles: Use cases, abstract models, and results. In *Proceedings of 7th International Workshop on Web Services and Formal Methods*, volume 6551 of *Lecture Notes in Computer Science*, 2010.
- [16] Karen Marie Lyng, Thomas Hildebrandt, and Raghava Rao Mukkamala. From paper based clinical practice guidelines to declarative workflow management. In *Proceedings ProHealth 08 workshop*, 2008.
- [17] Raghava Rao Mukkamala and Thomas Hildebrandt. From dynamic condition response structures to büchi automata. In *Proceedings of 4th IEEE International Symposium on Theoretical Aspects of Software Engineering (TASE 2010)*, August 2010.
- [18] Raghava Rao Mukkamala, Thomas Hildebrandt, and Janus Boris Tøth. The resultmaker online consultant: From declarative workflow management in practice to LTL. In *Proceeding of DDBP*, 2008.
- [19] Object Management Group BPMN Technical Committee. Business Process Model and Notation, version 2.0, 2010. <http://www.omg.org/spec/BPMN/2.0/>.
- [20] Organization for the Advancement of Structured Information Standards (OASIS). Web services human task (ws-humantask) specification, version 1.1, 2009. <http://docs.oasis-open.org/bpel4people/ws-humantask-1.1-spec-cd-06.pdf>.
- [21] Microsoft Research. Zing model checker. Webpage, 2010. <http://research.microsoft.com/en-us/projects/zing/>.
- [22] Pinar Senkul, Michael Kifer, and Ismail H. Toroslu. A logical framework for scheduling workflows under resource allocation constraints. In *In VLDB*, pages 694–705, 2002.
- [23] Munindar P. Singh, Greg Meredith, Christine Tomlinson, and Paul C. Attie. An event algebra for specifying and scheduling workflows. In *Proceedings of DASFAA*, pages 53–60. World Scientific Press, 1995.
- [24] Spin. On-the-fly, ltl model checking with spin. Webpage, 2008. <http://spinroot.com/spin/whatispin.html>.
- [25] Keith D. Swenson. *Mastering the Unpredictable: How Adaptive Case Management Will Revolutionize the Way That Knowledge Workers Get Things Done*. Meghan-Kiffer Press, 2010.
- [26] Wil M. P. van der Aalst and S Jablonski. Dealing with workflow change: Identification of issues and solutions. *International Journal of Computer Systems, Science, and Engineering*, 15(5):267–276, 2000.
- [27] Wil M. P. van der Aalst, Maja Pesic, and Helen Schonenberg. Declarative workflows: Balancing between flexibility and support. *Computer Science - R&D*, 23(2):99–113, 2009.
- [28] Wil M.P van der Aalst and Maja Pesic. A declarative approach for flexible business processes management. In *Proceedings DPM 2006*, LNCS. Springer Verlag, 2006.
- [29] W3C. Web services choreography description language, version 1.0, 2005. <http://www.w3.org/TR/ws-cdl-10/>.
- [30] Glynn Winskel. Event structures. In Wilfried Brauer, Wolfgang Reisig, and Grzegorz Rozenberg, editors, *Advances in Petri Nets*, volume 255 of *Lecture Notes in Computer Science*, pages 325–392. Springer, 1986.

- [31] Glynn Winskel and Mogens Nielsen. Models for concurrency. pages 1–148, 1995.
- [32] M. D. Zisman. *Representation, Specification and Automation of Office Procedures*. Philadelphia, Pa.: University of Pennsylvania, Wharton School, Department of Decision Sciences, Ph.D. Thesis, Sep, 1977.