Towards Trustworthy Adaptive Case Management with Dynamic Condition Response Graphs

Raghava Rao Mukkamala  
Postdoc, Process and System Models Group

Joint work with  
Thomas Hildebrandt  
Tijs Slaats  
IT University of Copenhagen, Denmark

(This work is supported by MT-LAB project)

EDOC 2013 Conference, Vancouver, Canada  
September 12, 2013
Road Map

• Part 1: Background & Motivation

• Part 2: DCR Graphs - Semantics

• Part 3: DCR Graphs for Adaptive Case Management

• Part 4: Conclusion & Future work
Adaptive Case Management (ACM)

Management of unstructured and dynamic processes, supporting knowledge workers rather than automation and efficiency.

A Spectrum Of Process Structure

Structured
• e.g., regulatory process

Structured with ad hoc exceptions
• e.g., financial back-office transactions

Adaptive with structured snippets
• e.g., insurance claims

Adaptive
• e.g., investigations

Adap7ve Case Management

• Unpredictable in their execution
• Goal oriented
• Emergent in nature and hence visibility and control can only be achieved during process execution
• Require the ad-hoc inclusion of new actors/events (run-time adaptation)
• Growing Knowledge Base

1) Kieth D Swenson: Mastering the unpredictable How Adaptive Case Management Will Revolutionize the Way That Knowledge Workers Get Things Done:
2) Nicolas Mundbrod, Jens Kolb and Manfred Reichert: Towards a System Support of Collaborative Knowledge Work
Adaptive Case Management (ACM)

- Examples of ACM Processes
  - Medicare Care, Emergency Medical Services
  - Collaborative Knowledge Work
  - Law Enforcement, Criminal Investigation
  - Complex Financial Transaction Services, Financial Audit
  - Crisis/Disaster Management, Floods, Natural calamities
Imperative versus Declarative

Imperative Models
- Explicit Control flow, over-specification, more rigid
- Focus is on *how* a set of tasks will be performed
- Easy to perceive and very successful for strictly procedural execution!

Classical trade-off between flexibility and support

Declarative Models
- Implicit Control, under specification, less rigid
- Focus is on *what* should be done instead of *how*
- Difficult to perceive, but suitable for modeling unstructured processes

Detailed Route Plan
(Imperative due to detailed specification)

GPS Application
(declarative due to under specification)
Part II

DCR Graphs
Dynamic Condition Response graphs (DCR graphs)


Labelled Prime Event Structures
[Winskel 1986]

Generalization

Process Matrix
Generalization and formalization

Dynamic Condition Response Graphs


IT UNIVERSITY OF COPENHAGEN
DCR Graphs - Overview

- Declarative workflow language
  - Formal semantics
  - Graphical Notation (Design + runtime)
  - developed as part of my PhD thesis

- Events and 5 relations/constraints
  - express all $\omega$-regular languages for infinite runs ($>\text{LTL}$)
- Generalized labeling function
DCR Graphs - Overview

- Operational Semantics
  - state represented as markings
  - execution based on transformation of markings - no state space explosion
  - easy to understand and represent graphically
  - supports run-time adaptation
  - Configurable initial state
- Support for Data, Time and Nesting as extensions
- Generalized Distribution technique
Related Work

• Declare (Pesic & van der Aalst 2006) & Linear-time Temporal Logic

• Guard-Stage-Milestone (GSM) Model (Hull et al 2010)

• Dynamic Changes in Workflow (Reichert and Dadam, 1997)

• Declarative configurable process specifications (Rychkova and Nurcan, 2011)

• State oriented view of business processes (Ilia Bider et. al. 2013)
DCR Graph - Formal Definition

A DCR Graph (DCR Graph) $G$ is a tuple $(E, M, \rightarrow\bullet, \bullet\rightarrow, \rightarrow\diamond, \rightarrow+, \rightarrow\%, L, l)$, where

(i) $E$ is the set of events (or activities),

(ii) $M = (Ex, Re, In) \in \mathcal{M}(G) = \text{def } \mathcal{P}(E) \times \mathcal{P}(E) \times \mathcal{P}(E)$ is marking,

(iii) $\rightarrow\bullet, \bullet\rightarrow, \rightarrow\diamond \subseteq E \times E$ are condition, response, milestone relations,

(iv) $\rightarrow+, \rightarrow\% \subseteq E \times E$ is the dynamic include relation and exclude relation, satisfying that $\forall e \in E. e \rightarrow+ \cap e \rightarrow\%= \emptyset$,

(v) $L$ is the set of labels,

(vi) $l : E \rightarrow \mathcal{P}(L)$ is a labeling function mapping events to sets of labels.
DCR Graphs - Example

Healthcare Workflow
• doctor has to sign after prescribing a medicine
• medicine can be given only after doctor’s sign
• If the nurse or pharmacist doesn’t trust the prescription, the doctor must look at it, change it if it is wrong, and sign again

prescribe

sign

give
don't trust
Healthcare Workflow
- doctor has to sign after prescribing a medicine
- medicine can be given only after doctor’s sign
- If the nurse or pharmacist doesn’t trust the prescription, the doctor must look at it, change it if it is wrong, and sign again
DCR Graphs - Example

Healthcare Workflow

- doctor has to sign after prescribing a medicine
- medicine can be given only after doctor’s sign
- If the nurse or pharmacist doesn’t trust the prescription, the doctor must look at it, change it if it is wrong, and sign again
DCR Graphs - Example

Healthcare Workflow
- doctor has to sign after prescribing a medicine
- medicine can be given only after doctor’s sign
- If the nurse or pharmacist doesn’t trust the prescription, the doctor must look at it, change it if it is wrong, and sign again
DCR Graph - Enabledness

For a DCR Graph $G = (E, M, \rightarrow\bullet, \rightarrow\diamond, \rightarrow+ , \rightarrow\% , L, I)$ with marking $M = \{\text{Ex, Re, In}\}$, we define that an event $e \in E$ is enabled, written as $M \vdash_G e$ if

i) $e \in \text{In}$

ii) $(\rightarrow\bullet e \cap \text{In}) \in \text{Ex}$

iii) $(\rightarrow\diamond e \cap \text{In}) \in E \setminus \text{Re}$
DCR Graph - Event Execution

For a DCR Graph $G = (E, M, \rightarrow\bullet, \bullet\rightarrow, \rightarrow\Diamond, \rightarrow+, \rightarrow\%, L, I)$ with marking $M = \{\text{Ex}, \text{Re}, \text{In}\}$ and with an enabled event $M \vdash_G e$, the result of executing the event $e$ will be a DCR Graph $G' = (E, M', \text{Act}, \rightarrow\bullet, \bullet\rightarrow, \rightarrow+, \rightarrow\%, \rightarrow\Diamond, L, I)$, where $M' = M \oplus_G e = \{\text{Ex}', \text{Re}', \text{In}'\}$ such that

i) $\text{Ex}' = \text{Ex} \cup \{e\}$

ii) $\text{Re}' = (\text{Re} \setminus \{e\}) \cup e \bullet\rightarrow$

iii) $\text{In}' = (\text{In} \cup e \rightarrow+) \setminus e \rightarrow\%$

---

### DCRG Marking (State)

<table>
<thead>
<tr>
<th>Executed (Ex)</th>
<th>Responses (Re)</th>
<th>Included (In)</th>
<th>Enabled Events</th>
</tr>
</thead>
<tbody>
<tr>
<td>prescribe</td>
<td>sign</td>
<td>prescribe</td>
<td>prescribe</td>
</tr>
<tr>
<td></td>
<td>give</td>
<td>sign</td>
<td></td>
</tr>
<tr>
<td></td>
<td>give</td>
<td>give</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>don't trust</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Executed (Ex)</th>
<th>Responses (Re)</th>
<th>Included (In)</th>
<th>Enabled Events</th>
</tr>
</thead>
<tbody>
<tr>
<td>prescribe</td>
<td>give</td>
<td>prescribe</td>
<td>prescribe</td>
</tr>
<tr>
<td>sign</td>
<td></td>
<td></td>
<td>sign</td>
</tr>
<tr>
<td>give</td>
<td></td>
<td></td>
<td>give</td>
</tr>
<tr>
<td>don't trust</td>
<td></td>
<td></td>
<td>don't trust</td>
</tr>
</tbody>
</table>

---

<prescribe>
For a DCR Graph $G = (E, M, →•, •→, →◊, →+, →%, L, l)$ an execution is a (finite or infinite) sequence of tuples $\{(M_i, e_i, a_i, M'_i)\}_{i \in [k]}$ such that

i) $M = M_0$, $\forall i \in [k].a_i \in l(e_i)$, $\forall i \in [k].M_i \vdash_G e_i$

ii) $\forall i \in [k].M'_i = M_i \oplus_G e_i$, $\forall i \in [k-1].M'_i = M_{i+1}$

Execution is accepting if $\forall i \in [k].(\forall e \in In_i \cap Re_i.\exists j \geq i.e_j = e \lor e \notin In'_j)$, where $M_i = (Ex_i, In_i, Re_i)$ and $M'_j = (Ex'_j, In'_j, Re'_j)$.

**DCRG Marking (State)**

<table>
<thead>
<tr>
<th>Executed (Ex)</th>
<th>Responses (Re)</th>
<th>Included (In)</th>
<th>Enabled Events</th>
</tr>
</thead>
<tbody>
<tr>
<td>prescribe</td>
<td>sign</td>
<td>prescribe</td>
<td>prescribe</td>
</tr>
<tr>
<td>sign</td>
<td>give</td>
<td>sign</td>
<td>sign</td>
</tr>
<tr>
<td>don’t trust</td>
<td>don’t trust</td>
<td>don’t trust</td>
<td>don’t trust</td>
</tr>
</tbody>
</table>

**Not Accepting**

**DCSG Marking (State)**

<table>
<thead>
<tr>
<th>Executed (Ex)</th>
<th>Responses (Re)</th>
<th>Included (In)</th>
<th>Enabled Events</th>
</tr>
</thead>
<tbody>
<tr>
<td>prescribe</td>
<td>give</td>
<td>prescribe</td>
<td>prescribe</td>
</tr>
<tr>
<td>sign</td>
<td>sign</td>
<td>sign</td>
<td>sign</td>
</tr>
<tr>
<td>don’t trust</td>
<td>don’t trust</td>
<td>don’t trust</td>
<td>give</td>
</tr>
</tbody>
</table>

**Not Accepting**
**DCR Graph - Execution**

For a DCR Graph \( G = (E, M, \rightarrow\bullet, \bullet\rightarrow, \rightarrow\circ, \rightarrow+, \rightarrow\%, L, l) \) an *execution* is a (finite or infinite) sequence of tuples \( \{(M_i, e_i, a_i, M'_i)\}_{i\in[k]} \) such that

i) \( M = M_0, \ \forall i \in [k].a_i \in l(e_i), \ \forall i \in [k].M_i \vdash_G e_i \)

ii) \( \forall i \in [k].M'_i = M_i \oplus_G e_i, \ \forall i \in [k-1].M'_i = M_{i+1} \)

Execution is *accepting* if \( \forall i \in [k].(\forall e \in In_i \cap Re_i, \exists j \geq i.e_j = e \lor e \notin In'_j) \), where \( M_i = (Ex_i, In_i, Re_i) \) and \( M'_j = (Ex'_j, In'_j, Re'_j) \).

**DCRG Marking (State)**

<table>
<thead>
<tr>
<th>Executed (Ex)</th>
<th>Responses (Re)</th>
<th>Included (In)</th>
</tr>
</thead>
<tbody>
<tr>
<td>prescribe</td>
<td>prescribe</td>
<td>sign</td>
</tr>
<tr>
<td>sign</td>
<td>sign</td>
<td>give</td>
</tr>
<tr>
<td>don't trust</td>
<td>give</td>
<td></td>
</tr>
</tbody>
</table>

**Enabled Events**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>prescribe</td>
<td>sign</td>
</tr>
<tr>
<td>sign</td>
<td>give</td>
</tr>
</tbody>
</table>

**Accepting**

<prescribe, sign, don’t trust, sign, give>
Formal Verification of DCR graph

- Formal Verification using Spin\(^1\) model checker
- DCR graphs automatically translated to PROMELA code
- Both safety and liveness properties can be verified
Formal verification of DCR graphs using SPIN

SPIN Model checker

Verification Tool

Graphical Editor + Simulator

Desktop Client

Web Client

Process Repository Service

XML Repository

DCRG workflow Engine

Process Execution Service

Subscription Service

Process Execution Notification Service

Runtime Monitor

DCRG Graphs Web Tool

Home
Process List
Process Management
Process Execution
About
Part III

Trustworthy Adaptive Case Management with DCR Graphs
Adaptive Operations

• Composition
• Event substitution
• Change operation
• Discard operation

ACM with DCR Graphs
Composition is a simple union of:

- events
- constraints between events
- markings (or runtime state)

Definition 4: Let $G_i = (E_i, M_i, \rightarrow_{i}, \bullet_{i}, \rightarrow_{\%i}, \rightarrow_{\rightarrow i}, \rightarrow_{\rightarrow\%i}, L_i, l_i)$, $M_i = (Ex_i, Re_i, In_i)$ for $i \in \{1, 2\}$. Then $G_1 \oplus G_2 = (E, M, \rightarrow, \bullet, \rightarrow\%, \rightarrow\rightarrow, \rightarrow\rightarrow\%, L, l)$, where

(i) $E = (E_1 \cup E_2)$
(ii) $M = (Ex_1 \cup Ex_2, Re_1 \cup Re_2, In_1 \cup In_2)$,
(iii) $\rightarrow\rightarrow = \rightarrow_{\rightarrow1} \cup \rightarrow_{\rightarrow2}$ for each $\rightarrow \in \{\rightarrow, \bullet \rightarrow, \rightarrow\%, \rightarrow\rightarrow\%\}$
(iv) $l(e) = l_1(e) \cup l_2(e)$ and $L = L_1 \cup L_2$
Composition

Composition is a simple union of:
- events
- constraints between events
- markings (or runtime state)

**Definition 4:** Let $G_i = (E_i, M_i, \rightarrow, \bullet, \rightarrow, \rightarrow, \rightarrow, \rightarrow, \rightarrow, \rightarrow, \rightarrow)$, $M_i = (Ex_i, Re_i, Ln_i)$ for $i \in \{1, 2\}$. Then

- $G_1 \oplus G_2 = (E, M, \rightarrow, \bullet, \rightarrow, \rightarrow, \rightarrow, \rightarrow, \rightarrow, \rightarrow, \rightarrow, \rightarrow)$, where
  - (i) $E = (E_1 \cup E_2)$
  - (ii) $M = (Ex_1 \cup Ex_2, Re_1 \cup Re_2, Ln_1 \cup Ln_2)$,
  - (iii) $\rightarrow = \rightarrow_1 \cup \rightarrow_2$ for each $\rightarrow \in \{\rightarrow, \bullet, \rightarrow, \rightarrow, \rightarrow, \rightarrow, \rightarrow\}$
  - (iv) $l(e) = l_1(e) \cup l_2(e)$ and $L = L_1 \cup L_2$
Composition

Composition is a simple union of
• events
• constraints between events
• markings (or runtime state)

Definition 4: Let \( G_i = (E_i, M_i, \rightarrow_i, \bullet_i, \rightarrow_i, \Rightarrow_i, \vdash_i, \rightarrow_+, \vdash_*, \vdash_%) \), \( M_i = (\text{Ex}_i, \text{Re}_i, \text{ln}_i) \) for \( i \in \{1, 2\} \). Then \( G_1 \oplus G_2 = (E, M, \rightarrow_+, \bullet, \rightarrow_*, \rightarrow_+, \vdash_*, \vdash_%, \vdash_+) \), where
(i) \( E = (E_1 \cup E_2) \)
(ii) \( M = (\text{Ex}_1 \cup \text{Ex}_2, \text{Re}_1 \cup \text{Re}_2, \text{ln}_1 \cup \text{ln}_2) \),
(iii) \( \rightarrow_\Rightarrow = \rightarrow_1 \cup \rightarrow_2 \) for each \( \rightarrow \in \{\rightarrow_+, \rightarrow_*, \rightarrow_+, \vdash_*\} \)
(iv) \( l(e) = l_1(e) \cup l_2(e) \) and \( L = L_1 \cup L_2 \)
Event Substitution Operation

Definition 5: Let \( G = (E, M, \rightarrow, \rightarrow, \rightarrow, \rightarrow, \rightarrow, \rightarrow) \), \( M = (Ex, Re, In) \) and \( e \in E, e' \in E \). The event substitution operation is defined as \( G[e \mapsto e'] = (E', M', \rightarrow, \rightarrow, \rightarrow, \rightarrow, \rightarrow, \rightarrow, \rightarrow) \) where

(i) \( E' = E \setminus \{e\} \cup \{e'\} \)

(ii) \( (e''[e \mapsto e'], a) \in l' \) if \( (e'', a) \in l \)

(iii) \( \forall \rightarrow \in \{\rightarrow, \rightarrow, \rightarrow, \rightarrow, \rightarrow, \rightarrow\}.e_1[e \mapsto e'] \rightarrow e_2[e \mapsto e'] \) if \( e_1 \rightarrow e_2 \)

(iv) \( M' = (Ex', Re', In') \) and \( \forall R \in \{Ex, Re, In\}.e''[e \mapsto e'] \in R' \) if \( e'' \in R \)

Event substitution

- new event will be substituted in event set
- labeling function will be updated
- all the constraint sets will be updated
- Finally, marking will be updated
Change Operation

Definition 6: Let \( G = (E, M, \rightarrow\bullet, \bullet\rightarrow, \rightarrow\otimes, \rightarrow+, \rightarrow\% \) , \( L, l \) ), \( M = (Ex, Re, In) \) and \( e \in E, e' \in E, A \subseteq L \). The change event operation is defined as \( G[e \mapsto (e', A)] = (E', M', \rightarrow\bullet', \bullet\rightarrow', \rightarrow\otimes', \rightarrow+', \rightarrow\%', L \cup A, l'') \) where

(i) \( G[e \mapsto e'] = (E', M', \rightarrow\bullet', \bullet\rightarrow', \rightarrow\otimes', \rightarrow+', \rightarrow\%', L, l') \)

(ii) \( l''(e'') = \begin{cases} A & \text{if } e'' = e \\ l'(e'') & \text{otherwise} \end{cases} \)

Change operation (for renaming of labels)
- Event substitution will be applied
- Labels are added to labels set
- Labeling function will be updated for new label mappings
Discard Operation

**Definition 7:** Let $G = (E, M, \rightarrow, \bullet \rightarrow, \rightarrow, \rightarrow, \rightarrow, \rightarrow, L, l)$ with $M = (Ex, Re, In)$. We define three *discard* operations by

(a) $G \otimes e = (E', M', \bullet \rightarrow', \bullet \rightarrow', \rightarrow', \rightarrow', \rightarrow', \rightarrow, \rightarrow, \rightarrow, \rightarrow, L, l')$ where

(i) $E' = E \setminus \{e\}, l' = l \setminus \{(e, l(e))\}$
(ii) $\forall R \in \{Ex, Re, In\}. R' = R \setminus \{e\}$
(iii) $\forall \rightarrow \in \{\rightarrow, \rightarrow, \rightarrow, \rightarrow, \rightarrow, \rightarrow\}$. $\rightarrow' = \{((e, e'), (e', e), (e, e) | e' \in E\}$

(b) $G \otimes (e \rightarrow_c e') = (E, M, \bullet \rightarrow', \bullet \rightarrow', \rightarrow', \rightarrow', \rightarrow', \rightarrow')$ where $\rightarrow \in \{\rightarrow, \rightarrow, \rightarrow, \rightarrow, \rightarrow, \rightarrow\}$ and $\rightarrow' = \{\rightarrow \setminus \{(e, e')\} if \rightarrow_c \rightarrow \rightarrow' otherwise$

(c) $G \otimes (e, R) = (E, M', \bullet \rightarrow, \bullet \rightarrow, \rightarrow, \rightarrow, \rightarrow, \rightarrow, \rightarrow, \rightarrow, L, l)$ where $M'$ is the obtained by removing the event from a set $R \in \{Ex, Re, In\}$ in $M$. 

3 overloaded versions
- discard an event
  - removed from event set
  - label mapping will be deleted
  - removed from marking
  - constraint set will be updated
- discard a constraint
  - constraint will be removed
- discard an event from marking
  - will be removed from the given set of the marking
ACM Example

DCRG Marking (State)

<table>
<thead>
<tr>
<th>Executed (Ex)</th>
<th>Responses (Re)</th>
<th>Enabled Events</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>pres med</td>
<td>pres. med.</td>
</tr>
<tr>
<td></td>
<td>sign</td>
<td>order tests</td>
</tr>
<tr>
<td></td>
<td>give med</td>
<td></td>
</tr>
<tr>
<td></td>
<td>order tests</td>
<td></td>
</tr>
<tr>
<td></td>
<td>do tests</td>
<td></td>
</tr>
<tr>
<td></td>
<td>exam. resu.</td>
<td></td>
</tr>
</tbody>
</table>

DCRG Marking (State)

<table>
<thead>
<tr>
<th>Executed (Ex)</th>
<th>Responses (Re)</th>
<th>Enabled Events</th>
</tr>
</thead>
<tbody>
<tr>
<td>order tests</td>
<td>sign</td>
<td>pres med</td>
</tr>
<tr>
<td></td>
<td>exam. resu.</td>
<td>order tests</td>
</tr>
<tr>
<td></td>
<td>give med</td>
<td>do tests</td>
</tr>
<tr>
<td></td>
<td></td>
<td>exam. resu.</td>
</tr>
</tbody>
</table>
ACM Example

Live Lock

DCRG Marking (State)

<table>
<thead>
<tr>
<th>Executed (Ex)</th>
<th>Responses (Re)</th>
<th>Included (In)</th>
<th>Enabled Events</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>pres med</td>
<td></td>
<td>pres. med.</td>
</tr>
<tr>
<td></td>
<td>sign</td>
<td></td>
<td>order tests</td>
</tr>
<tr>
<td></td>
<td>give med</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>order tests</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>do tests</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>exam. resu.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

DCRG Marking (State)

<table>
<thead>
<tr>
<th>Executed (Ex)</th>
<th>Responses (Re)</th>
<th>Included (In)</th>
<th>Enabled Events</th>
</tr>
</thead>
<tbody>
<tr>
<td>order tests</td>
<td>sign</td>
<td>pres med</td>
<td></td>
</tr>
<tr>
<td></td>
<td>exam. resu</td>
<td></td>
<td>sign</td>
</tr>
<tr>
<td></td>
<td>give med</td>
<td></td>
<td>order tests</td>
</tr>
<tr>
<td></td>
<td>order tests</td>
<td></td>
<td>do tests</td>
</tr>
<tr>
<td></td>
<td>exam. resu.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Properties

Safety Properties

For a dynamic condition response graph $G = (E, M, \rightarrow\bullet, \bullet\rightarrow, \rightarrow\circ, \rightarrow+, \rightarrow\% , L, I)$ we define that $G$ is deadlock free, if

$$\forall M' = (Ex', In', Re') \in M_{M'\rightarrow\bullet} \cdot (\exists e \in E.M' \vdash_G e \vee (In' \cap Re' = \emptyset))$$

and strongly deadlock free, if

$$\forall M' = (Ex', In', Re') \in M_{M'\rightarrow\bullet} \cdot (\exists e \in Re'.M' \vdash_G e \vee (In' \cap Re' = \emptyset)).$$

Liveness Properties

For a dynamic condition response graph $G = (E, M, \rightarrow\bullet, \bullet\rightarrow, \rightarrow\circ, \rightarrow+, \rightarrow\% , L, I)$ we define that the DCR Graph is live, if

$$\forall M' \in M_{M'\rightarrow\bullet} . acc_{M'}(G) \neq \emptyset,$$

and strongly live, if

$$\forall M' \in M_{M'\rightarrow\bullet} . macc_{M'}(G) \neq \emptyset.$$
Adapting a DCR graph

G \{(\text{prescribe} \rightarrow \bullet \text{sign}), \\
\text{sign} \rightarrow \bullet \text{give}, \text{prescribe} \rightarrow \bullet \text{sign}\}

G[\text{sign} \leftrightarrow \text{sign order tests}]

Thomas Hildebrandt and Raghava Rao Mukkamala
IT University of Copenhagen, Denmark
Adapting a DCR graph

\[
\begin{align*}
&\begin{array}{c}
\text{examine results} \\
| \quad | \\
\text{order tests} \quad \text{prescribe medicine} \\
\end{array} \\
&\begin{array}{c}
\text{sign order tests} \\
| \quad | \\
\text{give medicine} \\
\end{array}
\end{align*}
\]

\[+\]

\[
\begin{align*}
&\begin{array}{c}
\text{examine results} \\
| \quad | \\
\text{order tests} \quad \text{prescribe medicine} \\
\end{array} \\
&\begin{array}{c}
\text{sign order tests} \\
| \quad | \\
\text{give medicine} \\
\end{array}
\end{align*}
\]

\[=\]

\[
\begin{align*}
&\begin{array}{c}
\text{examine results} \\
| \quad | \\
\text{order tests} \quad \text{prescribe medicine} \\
\end{array} \\
&\begin{array}{c}
\text{sign order tests} \\
| \quad | \\
\text{give medicine} \\
\end{array}
\end{align*}
\]

\[
\begin{align*}
&\begin{array}{c}
\text{examine results} \\
\end{array} \\
&\begin{array}{c}
\text{order tests} \\
\end{array}
\end{align*}
\]

\[+\]

\[
\begin{align*}
&\begin{array}{c}
\text{examine results} \\
| \quad | \\
\text{order tests} \quad \text{prescribe medicine} \\
\end{array} \\
&\begin{array}{c}
\text{sign order tests} \\
\end{array}
\end{align*}
\]

\[=\]

\[
\begin{align*}
&\begin{array}{c}
\text{examine results} \\
\end{array} \\
&\begin{array}{c}
\text{order tests} \\
\end{array}
\end{align*}
\]
Part IV

Conclusion & Future Work
Conclusion

• DCR Graphs is formal declarative workflow model with graphical notation

• Intuitive Operational semantics based on markings

• Formally defined adaptive operations to allow run-time adaption

• Verification technique to verify properties using SPIN model checker

• Suitable for Adaptive Case Management
Current and Future work

- Native DCR Graphs Verification: Using DCR Graphs as specification for both model and property (already implemented and used for teaching)
- Data-Centric DCR Graphs: constraints depending also on data
- Mapping to related models (GSM, State-oriented, CMMN, ...)

Thanks! & Questions?