Distributed Dynamic Condition Response Structures

A formal model for declarative workflows

Seminar at IIIT-Hyderabad
9 June, 2010

Raghava Rao Mukkamala

PhD student,

Advisor: **Thomas T Hildebrandt**
Programming, logic and Semantics Group
IT University of Copenhagen, Denmark

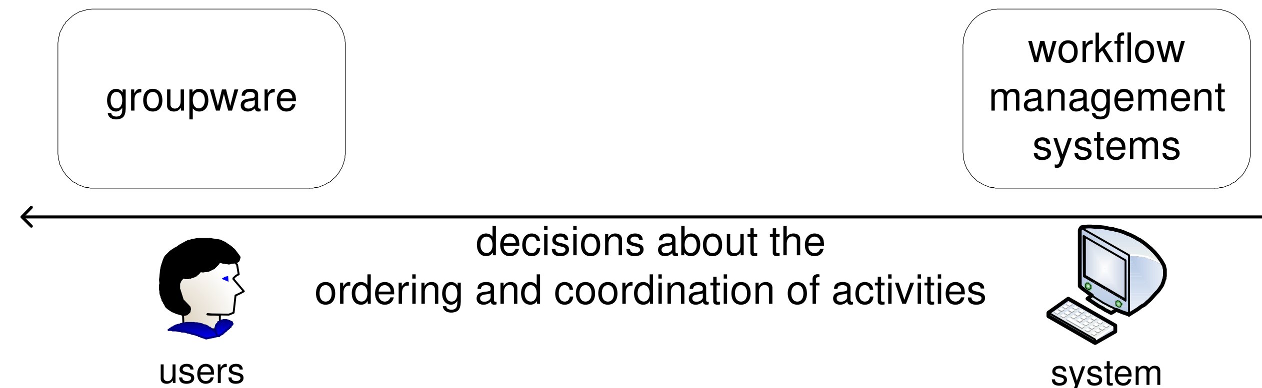
Overview

- Motivation
- PhD Project Overview
- Dynamic Condition Response Structures (DCRS)
  - DCRS graphical notation
  - Finite Runs
  - Infinite Runs
- Conclusion and Future Work

Motivation

Business Process Management systems

BPM systems support: collaboration, coordination and decision making



BPM Systems[1]

Groupware Systems

- Focus on supporting human collaboration and co-decision
- Ordering and Coordination of activities cannot be automated
- Users decide ordering and coordination of activities while executing the business process on the fly
- Examples: enhanced emails systems, group conferencing systems, document sharing systems

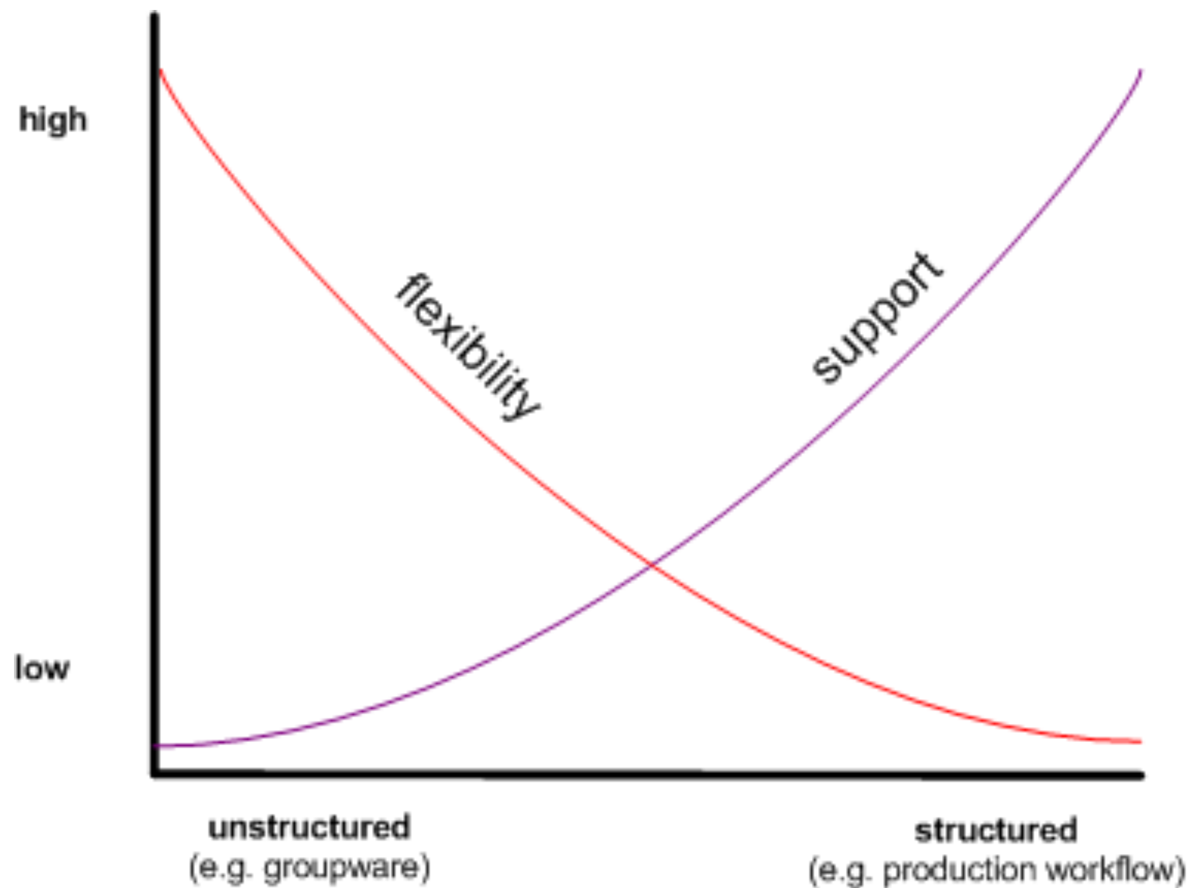
Workflow management Systems

- Focus on explicitly controlling ordering, coordination and execution of activities
- with little human intervention such entering necessary data
- Examples: Loan applications, Order handling systems, Case handling systems.

[1] Maja Pesic: Constraint-Based Workflow Management Systems: Shifting Control to Users (PhD Thesis)

Motivation

Flexibility versus Support in workflows



Classical trade-off between flexibility and support[1]

- Flexibility: ability to
  - defer: decide to decide later
  - change: decide to change model
  - deviate: decide to ignore model
- support: provide analysis and guidance
- unstructured: do what ever you want, but get no support (ex: email programs)
- structured: support, but no flexibility (traditional workflows)

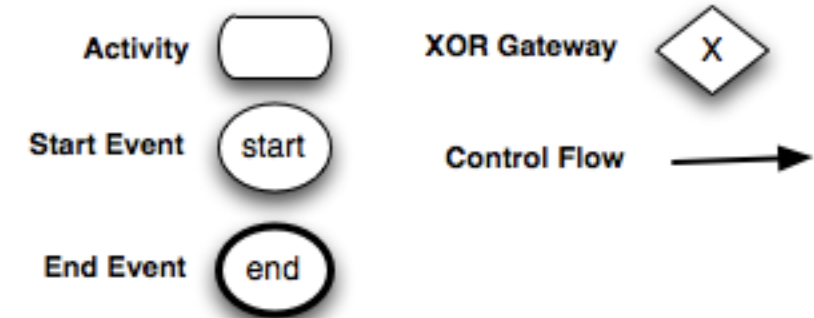
[1] W.M.P. van der Aalst et al. Declarative workflows: Balancing between flexibility and support

Motivation

Traditional workflow example in BPMN[1]

- Specification
  - ➡ 3 tasks: *bless*, *curse* and *pray*.
  - ➡ Rule: if you *curse* someone, then you MUST *pray* afterwards.

BPMN Symbols



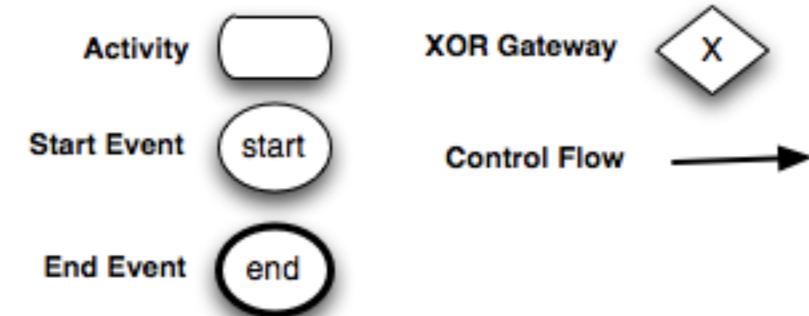
[1] BPMN: Business Process Modeling Notation

Motivation

Traditional workflow example in BPMN[1]

- Specification
  - ➡ 3 tasks: *bless*, *curse* and *pray*.
  - ➡ Rule: if you *curse* someone, then you MUST *pray* afterwards.

BPMN Symbols



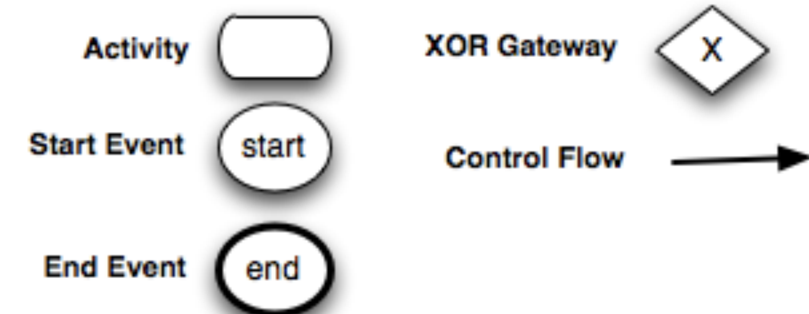
[1] BPMN: Business Process Modeling Notation

Motivation

Traditional workflow example in BPMN[1]

- Specification
  - ➡ 3 tasks: *bless*, *curse* and *pray*.
  - ➡ Rule: if you *curse* someone, then you MUST *pray* afterwards.

BPMN Symbols



[bless]

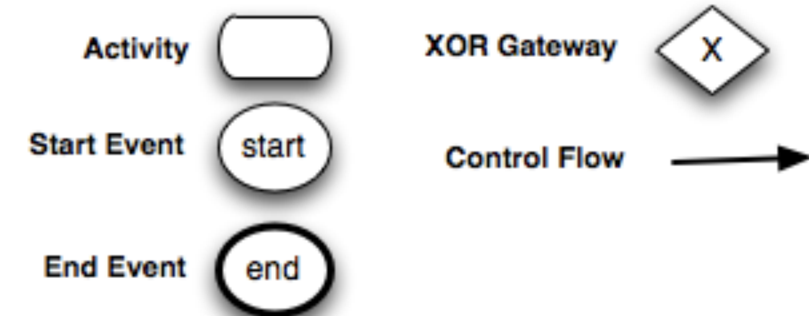
[1] BPMN: Business Process Modeling Notation

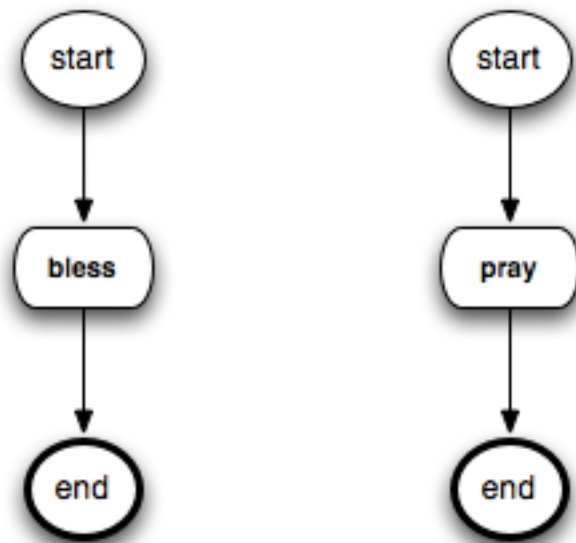
Motivation

Traditional workflow example in BPMN[1]

- Specification
  - ➡ 3 tasks: *bless*, *curse* and *pray*.
  - ➡ Rule: if you *curse* someone, then you MUST *pray* afterwards.

BPMN Symbols





[bless]

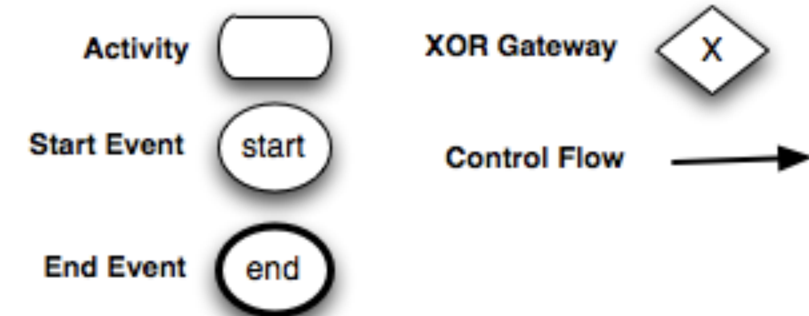
[1] BPMN: Business Process Modeling Notation

Motivation

Traditional workflow example in BPMN[1]

- Specification
  - ➡ 3 tasks: *bless*, *curse* and *pray*.
  - ➡ Rule: if you *curse* someone, then you MUST *pray* afterwards.

BPMN Symbols



[bless]

[pray]

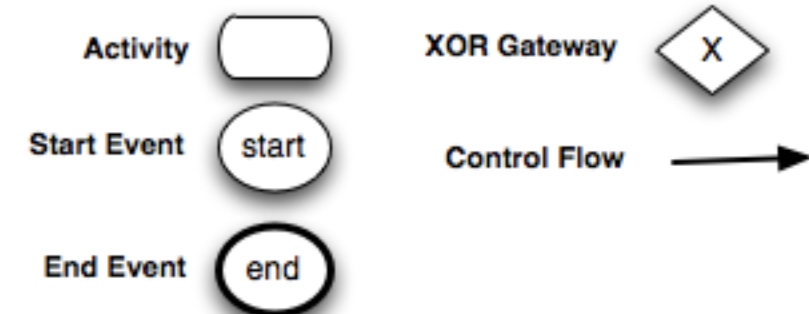
[1] BPMN: Business Process Modeling Notation

Motivation

Traditional workflow example in BPMN[1]

- Specification
  - ➡ 3 tasks: *bless*, *curse* and *pray*.
  - ➡ Rule: if you *curse* someone, then you MUST *pray* afterwards.

BPMN Symbols



[bless]

[pray]

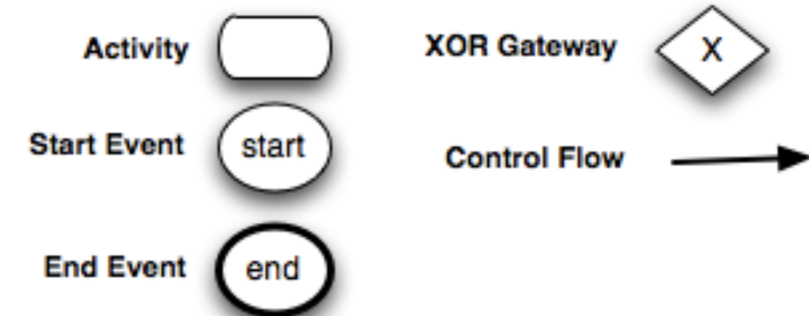
[1] BPMN: Business Process Modeling Notation

Motivation

Traditional workflow example in BPMN[1]

- Specification
  - ➡ 3 tasks: *bless*, *curse* and *pray*.
  - ➡ Rule: if you *curse* someone, then you MUST *pray* afterwards.

BPMN Symbols



[bless]

[pray]

[curse, pray]

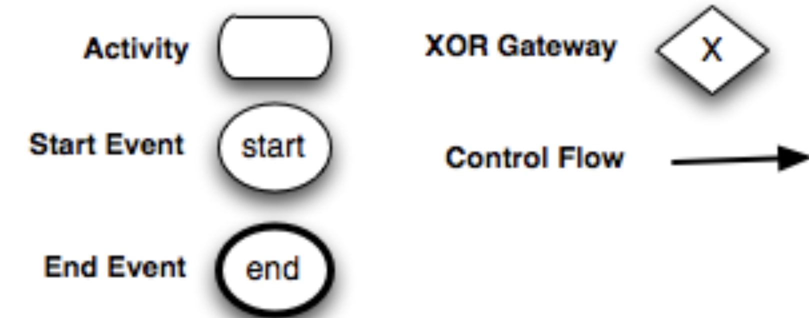
[1] BPMN: Business Process Modeling Notation

Motivation

Traditional workflow example in BPMN[1]

- Specification
  - ➡ 3 tasks: *bless*, *curse* and *pray*.
  - ➡ Rule: if you *curse* someone, then you MUST *pray* afterwards.

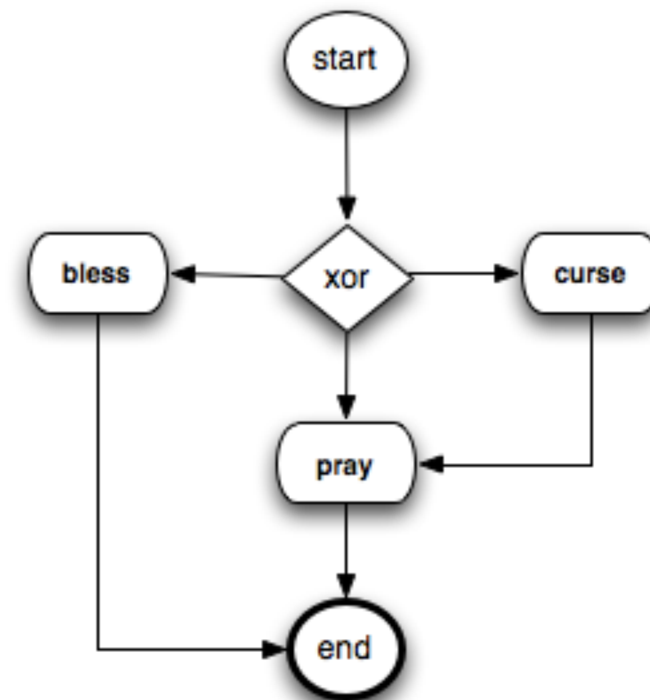
BPMN Symbols



[bless]

[pray]

[curse, pray]



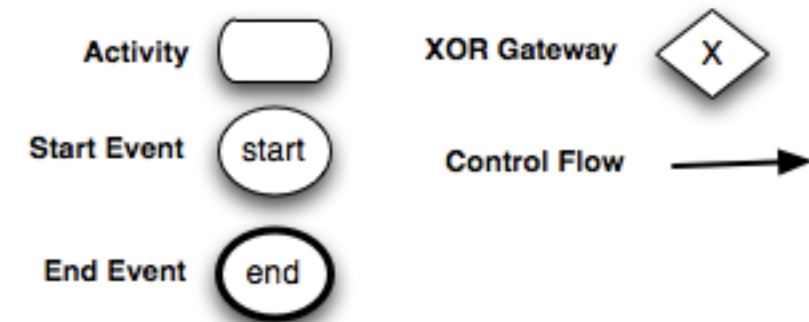
[1] BPMN: Business Process Modeling Notation

Motivation

Traditional workflow example in BPMN[1]

- Specification
  - ➡ 3 tasks: *bless*, *curse* and *pray*.
  - ➡ Rule: if you *curse* someone, then you MUST *pray* afterwards.

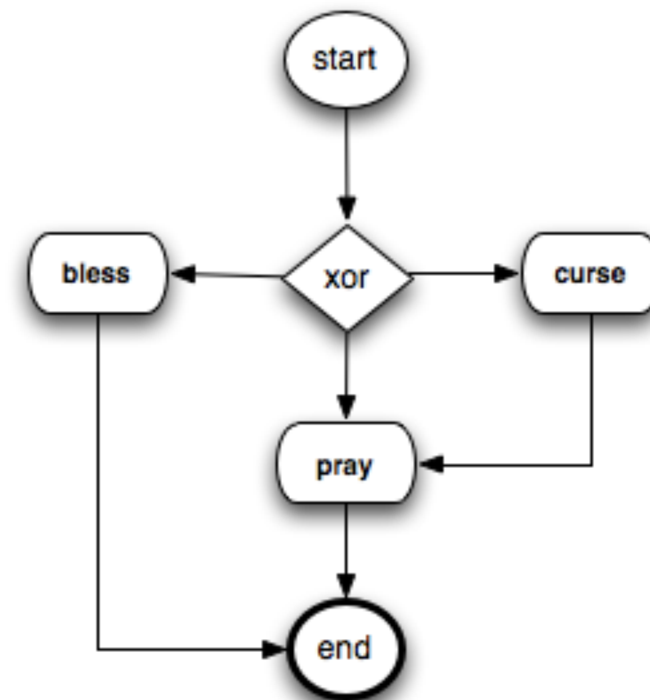
BPMN Symbols



[bless]

[pray]

[curse, pray]



[bless], [pray], [curse, pray]

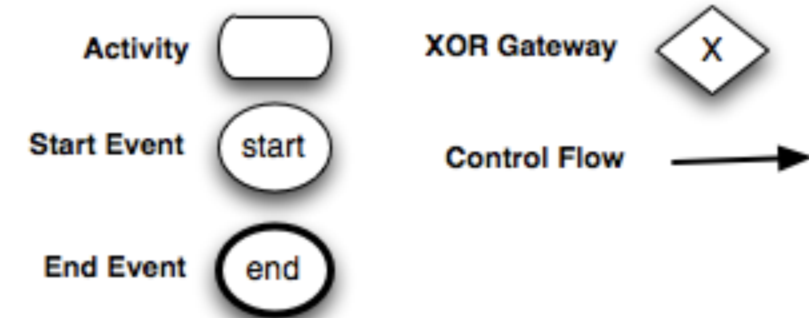
[1] BPMN: Business Process Modeling Notation

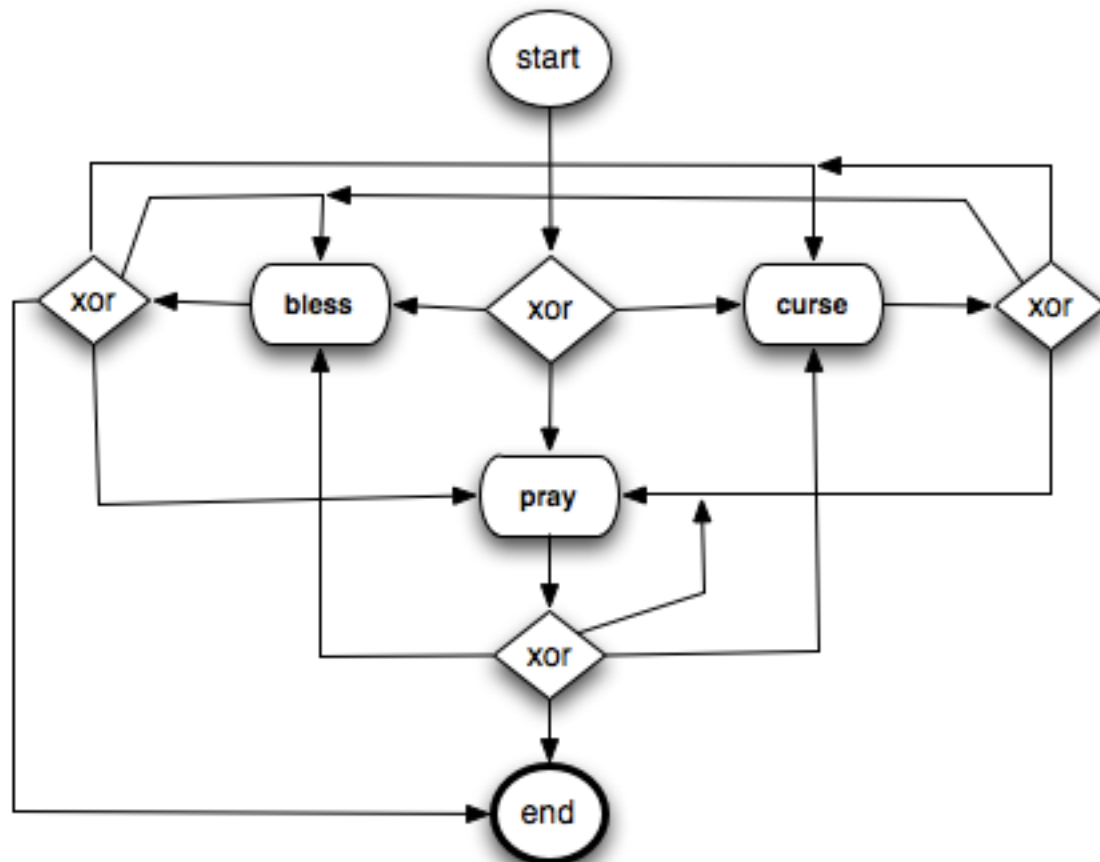
Workflow example in BPMN

Traditional workflow example in BPMN[1]

- Specification
  - ➡ 3 tasks: *bless*, *curse* and *pray*.
  - ➡ Rule: if you *curse* someone, then you MUST *pray* afterwards.

BPMN Symbols



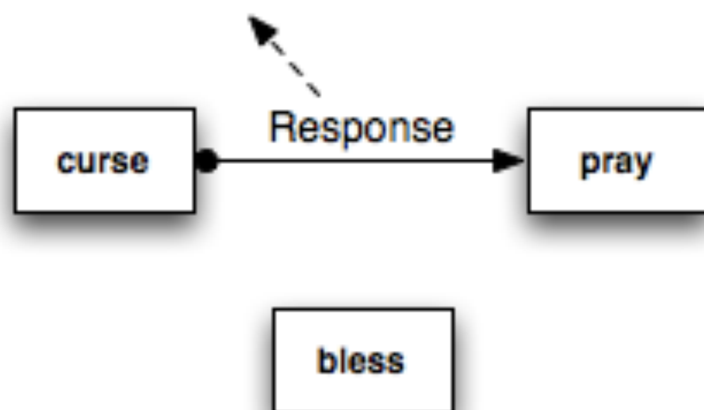


[bless, bless, curse, pray]
[curse, curse, pray, bless, bless]
[pray, bless, curse, bless, pray]
....

Workflow example using declarative modeling

Constraint-based Approach

- Specification
  - ➡ 3 tasks: *bless*, *curse* and *pray*.
  - ➡ Rule: if you *curse* someone, then you MUST *pray* afterwards.



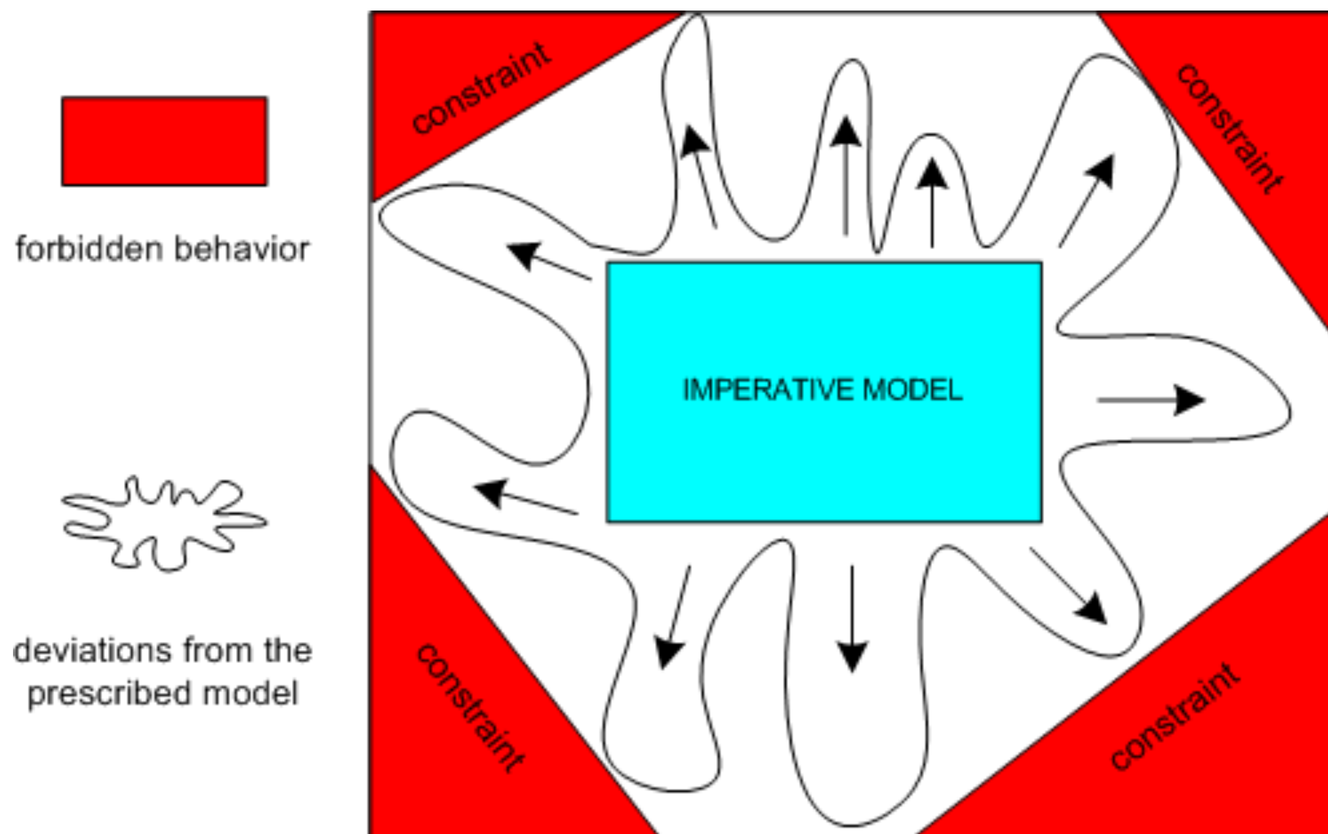
- Activities can be executed
  - ✓ any number of times
  - ✓ in any random order

  unless they are prevented by constraints
- Accepting run: any execution ending with accepting state, where all constraints are satisfied

Accepting runs:
[bless, bless]
[bless, bless, curse, pray]
[curse, curse, pray,]
[curse, curse, pray, bless, bless]

Non-accepting runs:
[pray, curse]
[bless, curse, pray, curse, bless]

Imperative versus Declarative models



Imperative versus declarative modeling languages[2]

- Imperative:
  - explicitly specify the control flow
  - over-constrain the control flow
  - adding a new constraint requires to make a new model
- Declarative:
  - implicitly specify the control flow
  - specify constrains to forbid the unwanted behavior
  - difficult to perceive what are the next possible actions
  - difficult to to get an overview about how to get to the end.

[2] M. Pesic and W.M.P. van der Aalst. A Declarative Approach for Flexible Business Processes Management

PhD Project Overview

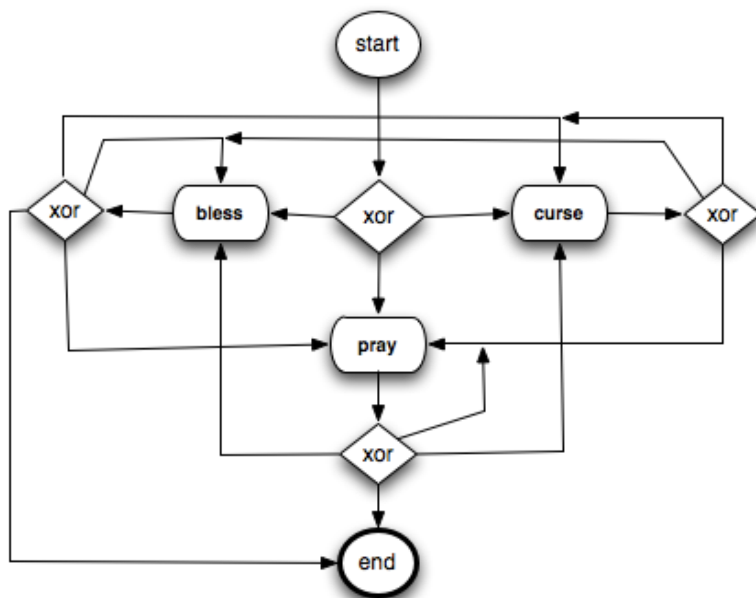
Title: Declarative Flexible Workflows for Trustworthy Pervasive Healthcare Services

Goal: To develop a formal process model for flexible workflows based on declarative approaches.

TrustCare Project: Trustworthy Pervasive Healthcare Services

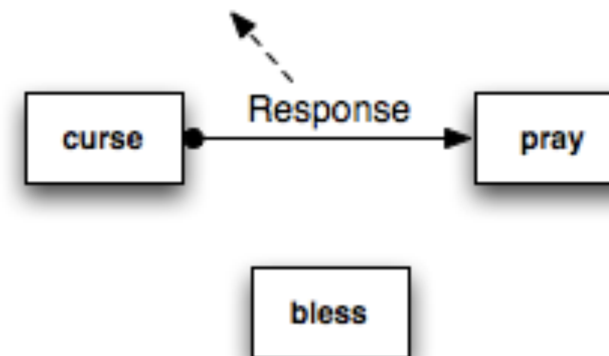
Research Methodology

Imperative Approach



Focus on **how** a set of tasks has to be performed

Declarative Approach



Focus on **what** should be done instead of **how**

Related Work

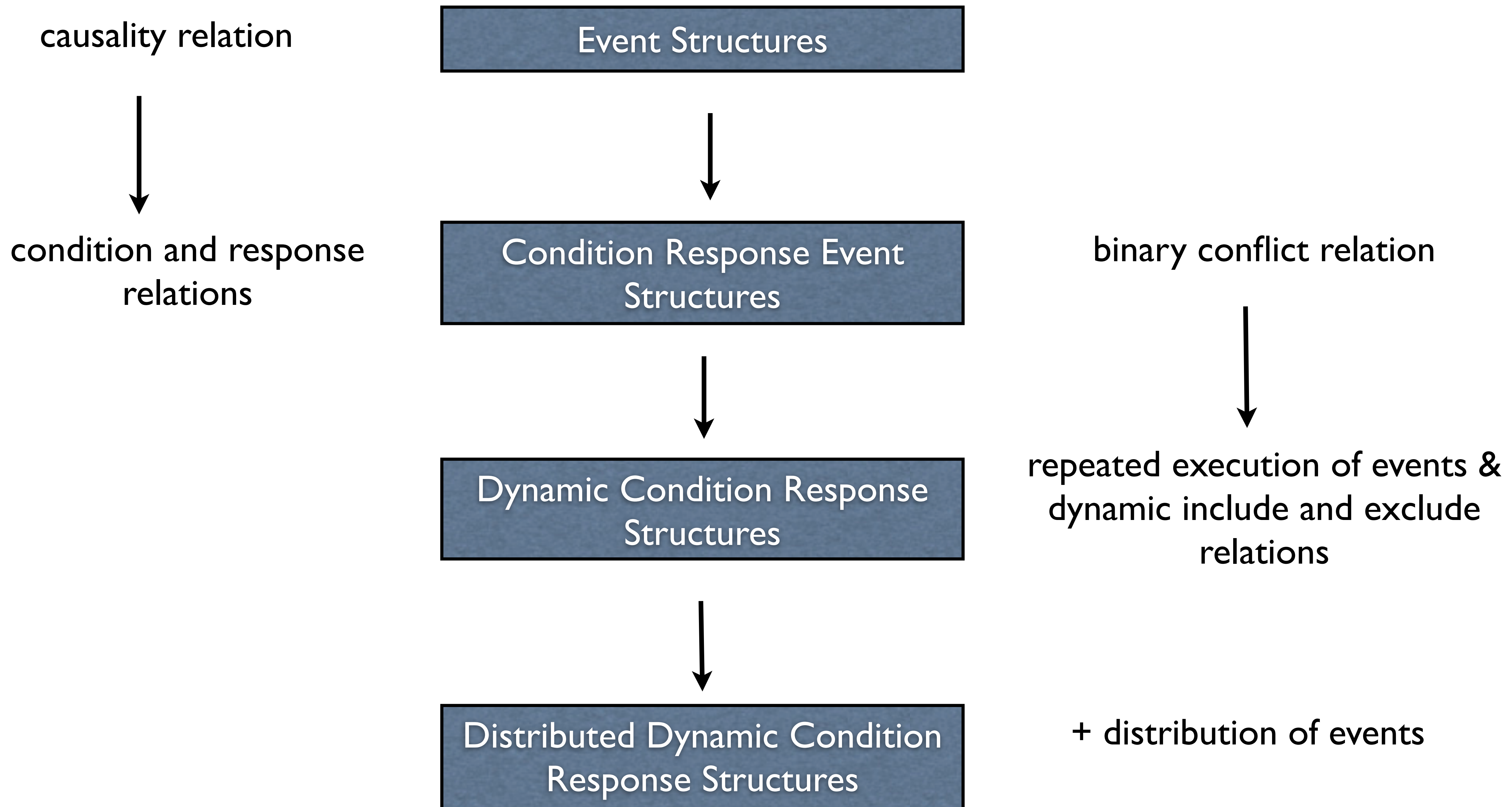
- Declare: declarative workflow language
    - W. M. P. van der Aalst, M. Pesic and H. Schonenberg: Declarative workflows: Balancing between flexibility and support. (2009).
    - W.M.P. van der Aalst and M. Pesic: DecSerFlow: Towards a Truly Declarative Service Flow Language. (2006)
- Event Calculus
    - Nihan Kesim Cicekli and Ilyas Cicekli. Formalizing the specification and execution of workflows using the event calculus. (2006)

Overview of Dynamic Condition Response Structures (DCRS)

- DCRS is formal (mathematical) process model for specification and execution of flexible workflows
- Using declarative approaches
- Based on Event Structures[3] which is a mathematical formalism for concurrent models
- DCRS has a graphical notation which can be used to specify workflows.
- Execution semantics
  - for finite runs mapped to labeled transition system
  - for infinite runs mapped to generalized Bu¨chi Automaton

[3] Glynn Winskel. Event structures. 1986.

Overview of Dynamic Condition Response Structures



Event Structures[3]

Definition

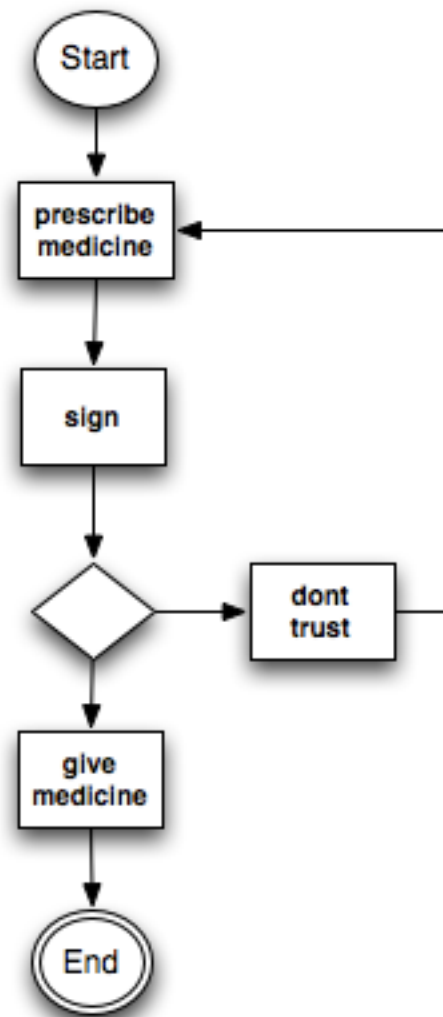
A labeled *prime event structure* (ES) over an alphabet *Act* is a 4-tuple $(E, \leq, \#, l)$ where

- E is a (possibly infinite) set of events
- $\leq \subseteq E \times E$ is the causality relation between events which is partial order
- $\# \subseteq E \times E$ is a binary conflict relation between events which is irreflexive and symmetric
- $l : E \rightarrow Act$ is the labeling function mapping events to actions

1. Causality relation satisfies principle of finite causes:
   $\forall e \in E : \{e' \in E \mid e' \leq e\}$ is finite.
2. Conflict relation satisfies principle of conflict heredity:
   $\forall e, e', e'' \in E. e \# e' \wedge e \leq e'' \Rightarrow e' \# e''$
3. A run is a sequence
   $e_0, e_1, ..e_n$ such that, $e_i \downarrow\leq \{e_0, ..e_{i-1}\}$ for $\forall i, j. \neg(e_i \# e_j)$

[3] Glynn Winskel. Event structures. 1986.

Health Care Workflow in Flow Chart



- doctor has to sign whenever he prescribes a medicine
- medicine can be given only after doctor's sign
- nurse can check medicine and say that "I don't trust medicine", in that case doctor has to either to sign again or change medicine + sign.

- possible runs
  - ✓ [pm, s, gm]
  - ✓ [pm, s, dt, pm, s, gm]
- missing
  - ◉ [pm, pm, s, gm]
  - ◉ [pm, s, pm, s, gm]

Health Care Workflow in Event Structures

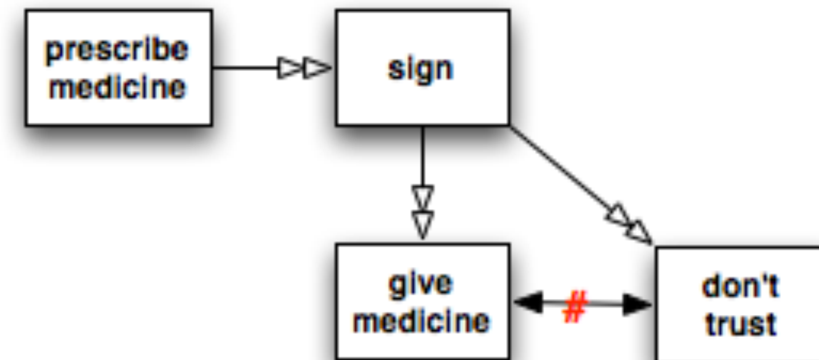
Events E = {pm, s, gm, dt}

Causality Relation : pm ≤ s ≤ gm, s ≤ dt

Conflict Relation: gm # dt

Possible Runs

- [pm, s, gm]
- [pm, s, dt]
- [pm, s]
- [pm]



Causality Relation

Conflict Relation

How can we make sure that every time when ***prescribe medicine*** happens eventually ***give medicine*** also happens?

- Events structures are missing
  - finite representation of infinite behavior
  - accepting condition
  - distribution of events

Condition Response Event Structures(CRES)

Definition

A labeled *condition response event structure* (CRES) over an alphabet *Act* is a tuple $(E, \leq_C, \leq_R, \#, l)$ where

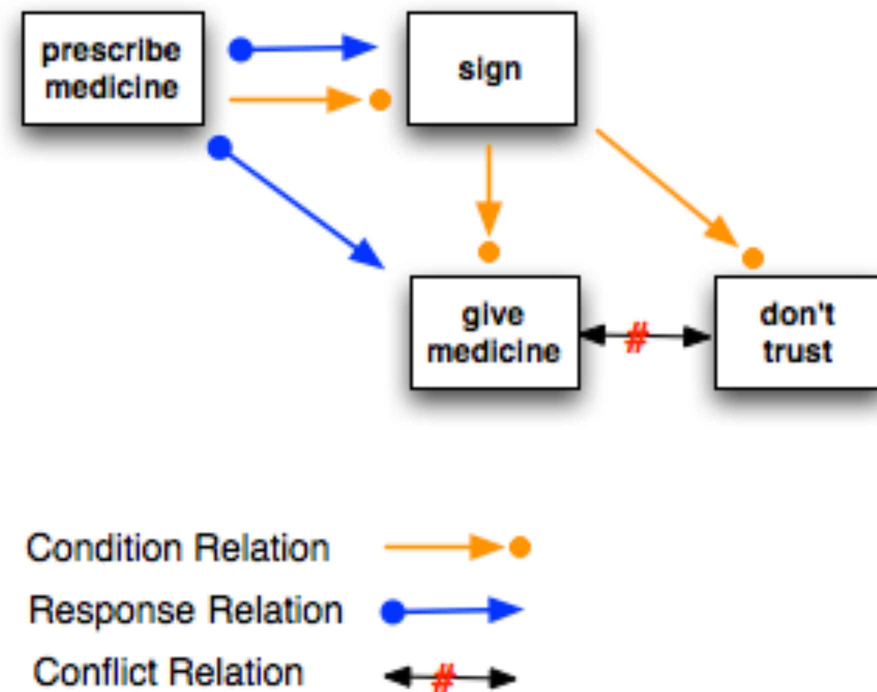
- $\leq_C \subseteq E \times E$ is the *condition* relation between events which is partial order
- $\leq_R \subseteq E \times E$ is the *response* relation between events, satisfying that $\leq = \leq_C \cup \leq_R$ is a acyclic relation
- $E, \#, l$ are same as Event Structures

Runs and accepting conditions

1. if e $\leq_C$e' then e must happen before e'
2. if e $\leq_R$e' then after e happens, e' must eventually happen or become in conflict for the computation to be accepting.

Condition Response Event Structures - 2

- $E = \{pm, s, gm, dt\}$
- $\leq_C = \{(pm, s), (s, gm), (s, dt)\}$
- $\leq_R = \{(pm, s), (pm, gm)\}$
- $\# = \{(gm, dt)\}$
- Possible runs
  - $[pm, s]$ OK, but not accepting
  - $[pm, s, gm]$ accepting
  - $[pm, s, dt]$ accepting



Proposition

The labelled prime event structure $(\mathrm{E}, \leq, \#, l, \mathrm{Act})$ *has the same runs as the accepting runs of the* CRES *structure (*E,Act,$\leq_C$,$\leq_R$,#,l,Act*) where* $\leq_C = \leq$, $\leq_R = \emptyset$

How would we able to re-execute ***prescribe medicine*** and/or ***sign*** after ***don't trust***?

Dynamic Condition Response Structures(DCRS)

Definition

A *dynamic condition response structure* (DCR) is a tuple
$D = (\text{E}, \text{Act}, \rightarrow\bullet, \bullet\rightarrow, \pm, l)$ where

- $\rightarrow\bullet \subseteq$ E$\times$ E is the *condition* relation
- $\bullet\rightarrow \subseteq$ E$\times$ E is the *response* relation
- $\pm : \text{E} \times \text{E} \rightarrow \{+, \%, *\}$ is the *dynamic inclusion/exclusion* relation.
- rest are same as Condition Response Event Structures

1. condition relation in DCRS ($\rightarrow\bullet$) is same as condition relation in CRES ($\leq_C$)
2. response relation in DCRS ($\bullet\rightarrow$) is same as response relation in CRES ($\leq_R$)
3. conflict relation is generalized to include relation + exclude relation to include/exclude events dynamically.
4. $\pm(e, e') = +$ if event e gets executed, then it will include event e'
5. $\pm(e, e') = \%$ if event e gets executed, then it will exclude event e'
6. conflict relation is monotone (once an event is in conflict it stays in conflict), but dynamic inclusion/exclusion allows an event to alternate between being in conflict and not.
7. execution of an event only depends on the condition relation restricted to the currently included events.

Dynamic Condition Response Structures(DCRS)

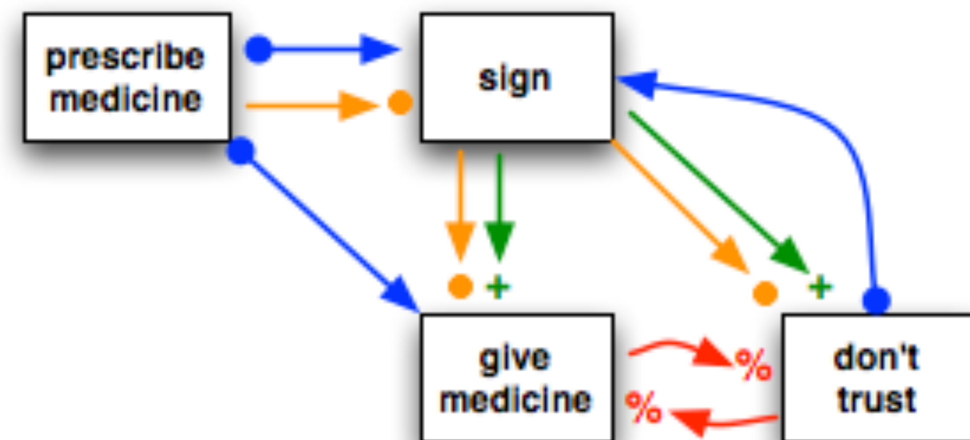
Proposition

The condition response event structure $(\mathrm{E}, \leq_C, \leq_R, \#, l, \mathrm{Act})$ *has the same accepting runs as the accepting runs of the* DCR *structure* $(\mathrm{E}, \mathrm{Act}, \rightarrow\bullet, \bullet\rightarrow, \pm, l)$ *where* $\rightarrow\bullet = \leq_C$, $\bullet\rightarrow = \leq_R$, $\forall e, e' \in \mathrm{E}.\ \pm(e, e') = \%$ *if* $e = e'$ *or* $e \# e'$ *and otherwise* $\pm(e, e') = *$.

relation in CRES

Encoding of # in DCRS

- $E = \{pm, s, gm, dt\}$
- $\rightarrow\bullet = \{(pm, s), (s, gm), (s, dt)\}$
- $\bullet\rightarrow = \{(pm, s), (pm, gm), (dt, ss)\}$
- $\rightarrow+ = \{(s, gm), (s, dt)\}$
- $\rightarrow\% = \{(gm, dt), (dt, gm)\}$
- Possible runs
  - $[pm, s, dt, s, gm]$
  - $[pm, s, dt, pm, s, gm]$
  - $[pm, s, pm, s, gm]$



Distributed Dynamic Condition Response Structures*

Definition

A *distributed* dynamic condition response structure(DDCR) is a tuple

$$(E, Act, \rightarrow\bullet, \bullet\rightarrow, \pm, l, R, P, as)$$

where $(E, Act, \rightarrow\bullet, \bullet\rightarrow, \pm, l)$ is a dynamic condition response structure, R is a set of *roles*, P is a set of *principals* (e.g. persons/processors/agents) and $as \subseteq (P \cup Act) \times R$ is the role assignment relation to executors and actions.

- assigning roles to actions provide granularity of permissions
- assigning principals to roles gives the permission to execute actions

[*] Thomas Hildebrandt and Raghava Rao Mukkamala. Distributed dynamic condition response structures. Workshop on Programming Language Approaches to Concurrency and Communication-cEntric Software (PLACES 10), Cyprus, March 2010.

Execution semantics:(strong) Accepting Condition for Finite Runs

Definition

For a distributed DCR $D = (\mathsf{E}, \mathsf{Act}, \rightarrow\bullet, \bullet\rightarrow, \pm, l, \mathsf{R}, \mathsf{P}, \mathsf{as})$ the corresponding labelled transition systems $T(D)$ to be the tuple $(S, (\emptyset, \mathsf{E}, \emptyset), \rightarrow \subseteq S \times \mathsf{Act} \times S)$ where $S = \mathcal{P}(\mathsf{E}) \times \mathcal{P}(\mathsf{E}) \times \mathcal{P}(\mathsf{E})$ is the set of states, $(\emptyset, \mathsf{E}, \emptyset) \in S$ is the initial state, $\rightarrow \subseteq S \times (\mathsf{P} \times \mathsf{Act} \times \mathsf{R}) \times S$ is the transition relation given by

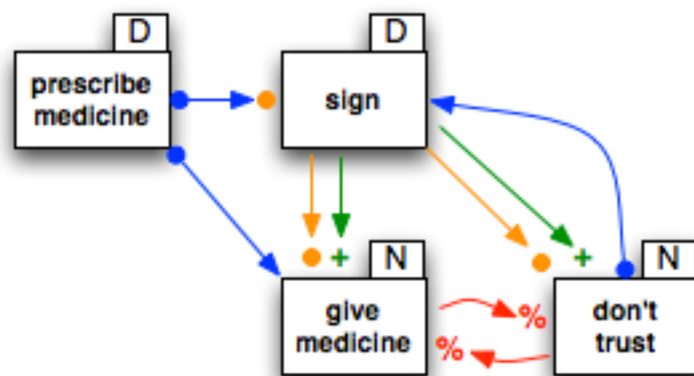
$$(E, I, R) \xrightarrow{(e,(p,a,r))} (E \cup \{e\}, I', R') \text{ where}$$

- $e \in I$, $l(e) = a$, p as r, and a as r
- $\{e' \in I \mid e' \rightarrow\bullet\ e\} \subseteq E$
- $I' = (I \cup \{e' \mid \pm(e, e') = +\}) \setminus \{e' \mid \pm(e, e') = \%\}$
- $R' = (R \setminus \{e\}) \cup \{e' \mid e \bullet\rightarrow e'\}$

- map to a labelled transition system to defined accepting runs
- states of transition system will be (E, I, R) where $E \subseteq \mathsf{E}$ is set of happened events, $I \subseteq \mathsf{E}$ represents set of currently included events, $R \subseteq \mathsf{E}$ represents set of pending response events
- first condition says that only currently include events can be executed, the label (p, a, r) says that the label of event e must be a, which must be assigned to a role r and principal p
- second condition says that all condition events to e must have been executed
- third and fourth conditions are updates to sets of included and pending responses.
- Accepting State: Any state with no pending responses ($R \cap I = \emptyset$)
- Accepting run: Any run which is ending with accepting state ($R \cap I = \emptyset$)

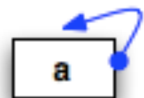
Health Care Workflow in DCRS

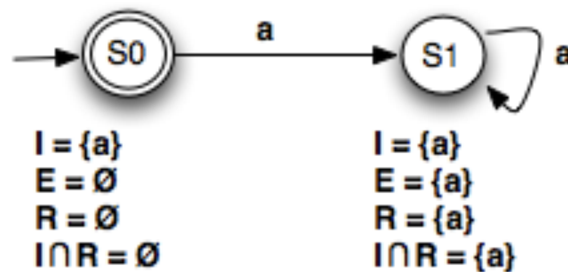
- doctor has to sign whenever he prescribes a medicine
- medicine can be given only after doctor's sign
- nurse can check medicine and say that "I don't trust medicine", in that case doctor has to either to sign again or change medicine + sign.

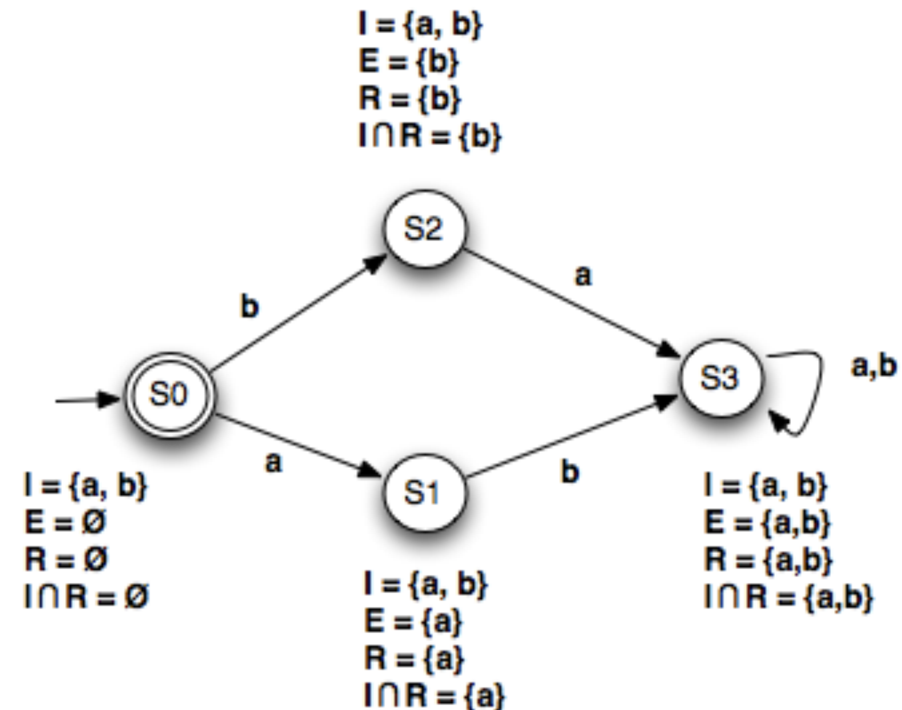


- OK and Accepting
  - ✓ [pm, s, gm, gm]
  - ✓ [pm, s, dt, s, gm]
  - ✓ [pm, s, dt, pm, s, gm]
  - ✓ [pm, pm, s, gm]
  - ✓ [pm, s, pm, s, gm]
- Not Possible
  - ◉ [pm, gm]
  - ◉ [pm, s, dt, gm]
- Possible, but not Accepting
  - ◉ [pm, s, gm, pm] (pending : gm, s)
  - ◉ [pm, s, dt] (pending : s)

DCRS for Infinite Runs - Motivation







- In the first example a run a^{ω} should be accepting.
- In the second example, a run $a^{*}\ b^{*}\ (ab)^{\omega}$ should be accepting.
- But strong accepting condition ($R \cap I = \varnothing$) does not allow those runs accepting.

(Weaker) Accepting Condition for infinite runs*

For a finite distributed DCR $D = (\mathrm{E}, \mathrm{Act}, \rightarrow\!\bullet, \bullet\!\rightarrow, \pm, l, \mathrm{R}, \mathrm{P}, as)$ where $\mathrm{E} = \{e_1, \ldots, e_n\}$ we define the corresponding Büchi-automaton $Aut(D)$ to be the tuple $(S, s, \rightarrow \subseteq S \times \mathrm{Act} \times S, F)$ where $S = \mathcal{P}(\mathrm{E}) \times \mathcal{P}(\mathrm{E}) \times \mathcal{P}(\mathrm{E}) \times \{1, \ldots, n\} \times \{0, 1\}$ is the set of states and $s = (\emptyset, \mathrm{E}, \emptyset, 1, 1) \in S$ is the initial state and $F = \mathcal{P}(\mathrm{E}) \times \mathcal{P}(\mathrm{E}) \times \mathcal{P}(\mathrm{E}) \times \{1, \ldots, n\} \times \{1\}$ is the set of final or accepting states. $\rightarrow \subseteq S \times (\mathrm{P} \times \mathrm{Act} \times \mathrm{R}) \times S$ is the transition relation given by

$$(E, I, R, i, j) \xrightarrow{(p,a,r)} (E \cup \{e\}, I', R', i', j') \text{ where}$$

- Semantics of E, I, R are same as previous accepting condition
- $j' = 1$ if
  1. $I' \cap R' = \emptyset$
  2. $min(M) \in (I \cap R \backslash (I' \cap R')) \cup \{e\}$
  3. $M = \emptyset$ and $min(I \cap R) \in (I \cap R \backslash (I' \cap R')) \cup \{e\}$

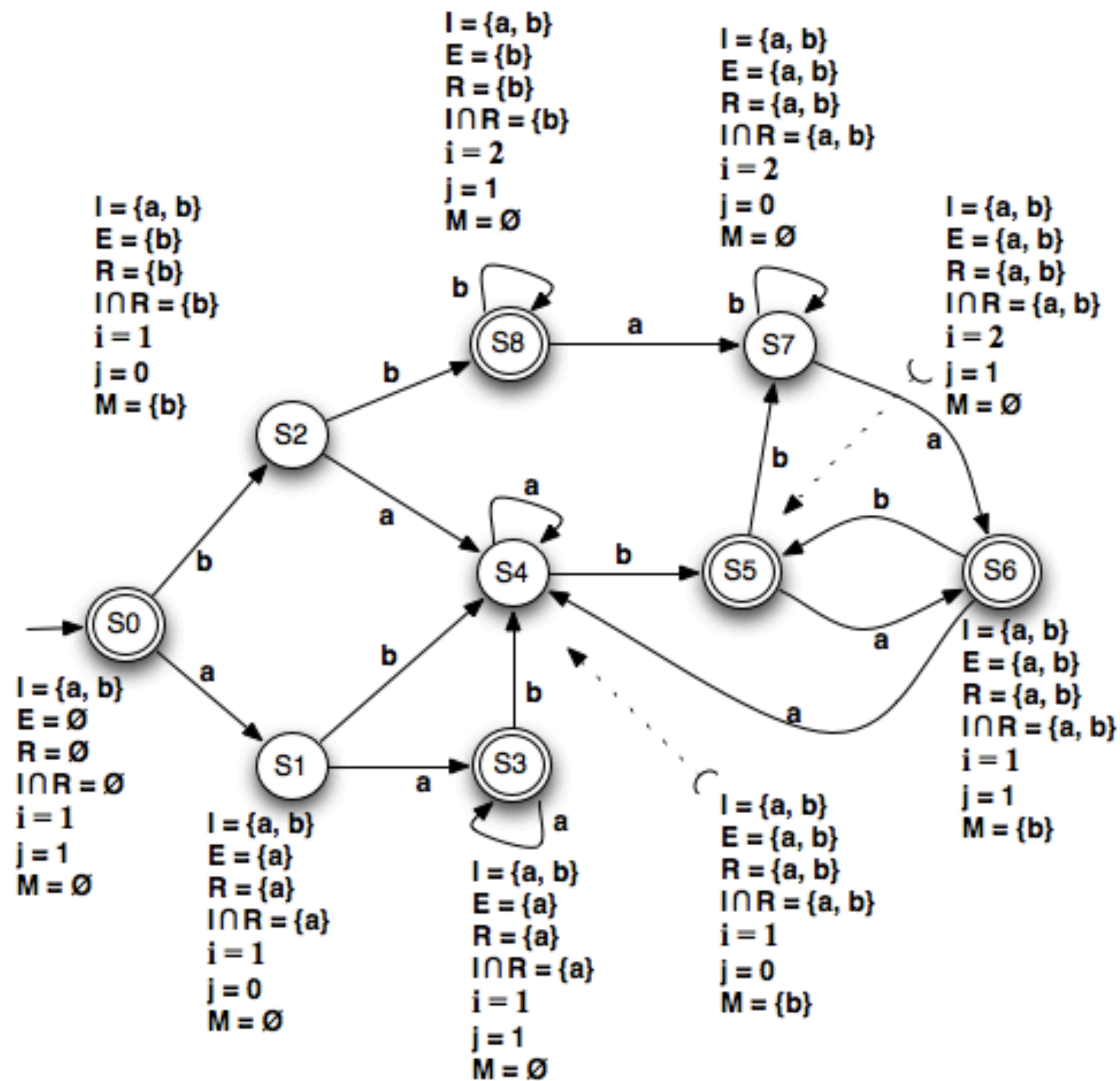
  otherwise $j' = 0$.
- $i' = rank(min(M))$ if $min(M) \in (I \cap R \backslash (I' \cap R')) \cup \{e\}$
- $i' = rank(I \cap R)$ if $M = \emptyset$ and $min(I \cap R) \in (I \cap R \backslash (I' \cap R')) \cup \{e\}$
- $i' = i$ otherwise.

for $M = \{e \in I \cap R \mid rank(e) > i\}$.

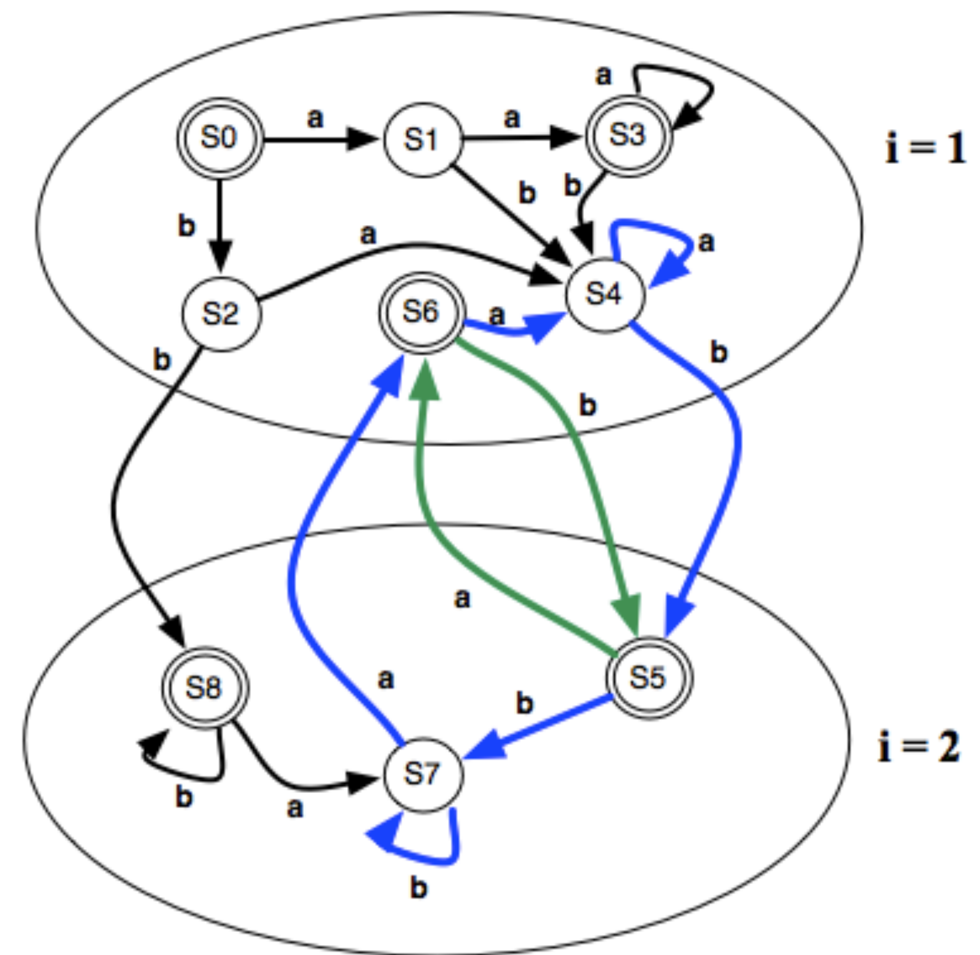
- Mapped to a Generalized Buchi automata
- Instead of looking for $R \cap I = \emptyset$, we make sure that all pending response events ($R \cap I$) will be eventually executed infinitely often.
- In other words, infinite run is not accepting if one or more pending response event(s) will never executed.

[*] Raghava Rao Mukkamala and Thomas Hildebrandt. From Dynamic Condition Response Structures to Büchi Automata. 4th IEEE International Symposium on Theoretical Aspects of Software Engineering (TASE 2010), Taiwan, August 2010.

Accepting condition for infinite runs - example



Infinite runs: Bu¨chi Automaton for example



Accepting run for case where both a and b are executed with i copies of state

Conclusion and Future Work

Conclusion

- We have developed a formal process model which is
  - ✓ more flexible than traditional workflows
  - ✓ easy to understand and to execute
  - ✓ suitable for applying model checking and verification using SPIN tool.

Future work

- Currently developing property checker for DCRS model in Microsoft Research, Bangalore.
- Support for Data, Time, Sub-processes, Exceptions and Compensation
- Quantitative and probabilistic modalities on constraints
- A prototype engine for DCRS model

Thank You
Questions & Comments ?