

Of Business Process Modeling with BPMN and Situation Calculus

Sylvain Bouveret

Onera – DTIM

Onera, Département Traitement de l'Information et Modélisation



What is process modeling?

Process modeling aims at representing in a single framework processes of the same type.



What is process modeling ?

Process modeling aims at representing in a single framework processes of the same type.





Introduction

What is process modeling ?

Process modeling aims at representing in a single framework processes of the same type.





Introduction

What is process modeling ?

Process modeling aims at representing in a single framework processes of the same type.

Accueil

PROCESSUS R2

km, 23/03/2009 - 14:34

Intitulé :
Conclure les travaux

Le processus assure la réalisation des travaux de recherche de la proposition à la clôture.

Date de la publication :
01/10/2008

Garant :
François Châtoneau

Liste des pilotes :

DAMP : B. Hailard	DADS : J. Legrain-Naudin	DAFE : F. Bouvier	DCSD : N. Zinbert
DEFA : D. Desormes	DEPR : S. Attia	DESP :	DMRE : H. Gellie
DMR : F. Dayzac	DMSC : C. Hachette	DMSM : M. Thomas	DOTA : M.-T. Velkue
DMRS : J. Bourrely	DGNA : A.-M. Vallic	DTM : B. Lécussat	

Fiche d'identification du processus :
R2_Fiche.pdf

Convales rendus de revue de processus :
r2-revue-070607.pdf
r2-revue-080320.pdf
R2-RevProc-08-09-13-CRM.pdf

Indicateurs de processus:



What is process modeling ?

Process modeling aims at representing in a single framework processes of the same type.

It can be :

- **descriptive** (what happens during the process ?);
- **prescriptive** (how can we build processes that satisfy some criteria ?);
- **explanatory** (why did the process end like that ?).



What is process modeling ?

Five main objectives [Curtis et al., 1992] :

- 1 facilitation of the communication and understanding between individuals ;
- 2 process improvement ;
- 3 project management ;
- 4 process monitoring ;
- 5 automatic execution (simulation).



Curtis, B., Kellner, M. I., and Over, J. (1992).

Process modeling.

Commun. ACM, 35(9) :75–90.



Process modeling : main concepts

- **Process** : a set of partially ordered steps aiming at satisfying a **goal**.
- **Process step** : atomic action of a process that has no visible sub-structure. It is assigned to a **role**, and manipulates **artifacts**.
- **Agent** : actor (individual, organization...) that participates to the process.
- **Role** : consistent set of process elements, assigned to an agent as a **functional responsibility**.
- **Artifact / resource** : product manipulated by the process (created, needed or modified).



Curtis, B., Kellner, M. I., and Over, J. (1992).

Process modeling.

Commun. ACM, 35(9) :75–90.



Feller, P. and Humphrey, W. (1992).

Software process development and enactment : Concepts and definitions.

Technical report, Software Engineering Institute, Pittsburgh.



Existing approach and tools

- In the ancient time, free text and very simple diagrams ;
- then, (imperative) programming languages ;
- from the 90s, conception and system analysis tools, AI, discrete event languages, statecharts, Petri nets, flow control diagrams, functional languages, object oriented modeling, PERT diagrams,...

Some recent modeling languages :

- *Business Process Modeling Notation* [Object Management Group, 2008] : a standard graphical language by the OMG.
- *Web Services Business Process Execution Language* : an XML language by the OASIS.
- *Software Process Engineering Metamodel* : another metamodel by the OMG.



Object Management Group (2008).

Business process modeling notation, v1.1.

Technical report, Object Management Group.



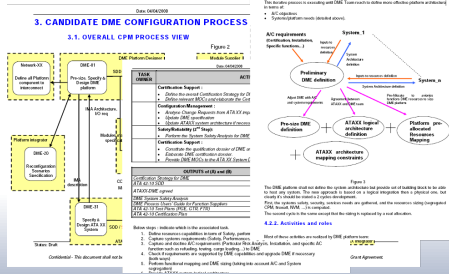
Context of the study

- **Project SCARLETT** : FP7 project, addresses the new generation of Integrated Modular Avionics, including reconfiguration features.
- What about business process modeling?



Context of the study

- **Project SCARLETT** : FP7 project, addresses the new generation of Integrated Modular Avionics, including reconfiguration features.
- What about business process modeling?

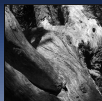


Objective : study, understand and analyze the DME development process.



What's at the menu today?

- 1 **A short introduction to Business Process Modeling Notation**
 - BPMN: a snapshot
 - Core modeling elements: actors, flow objects, connectives and artifacts
 - Examples
 - Conclusion about BPMN
- 2 **Situation Calculus**
 - Situation Calculus in a Nutshell
 - The language of the situation calculus
 - Example
 - Extensions
- 3 **From BPMN to Situation Calculus**
 - Principles of the transcription
 - Examples
- 4 **Conclusion**



What is BPMN ?

- A graphical notation for specifying **business process**.
- Standardized (v1.1) and currently maintained by the **Object Management Group** [Object Management Group, 2008]
- Based on the concept of **workflow**.
- Core modeling elements : **pools, activities, events, gateways**.
- Data and information exchanged can be modeled using **artifacts** and **messages**.



Object Management Group (2008).

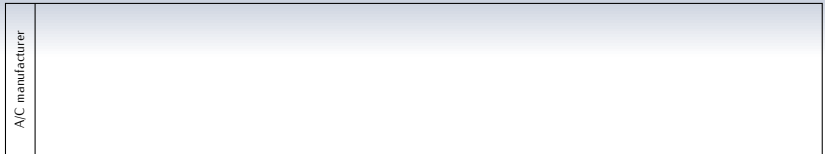
Business process modeling notation, v1.1.

Technical report, Object Management Group.

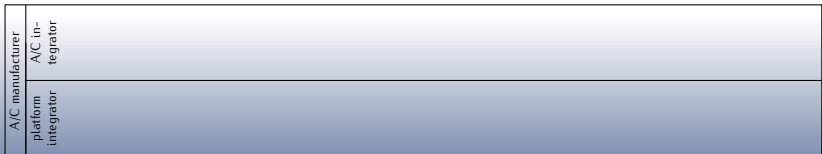


Pools and Lanes

- A **pool** : an actor / a role.



- A **lane** : a “sub-actor” (a way to partition roles into sub-roles)



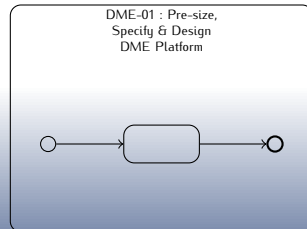


Flow objects : activities

- A **task** : some **atomic** activity that is performed during the process.



- A **subprocess** : a **compound** activity that can be refined into a finer level of details.






Flow objects : events

- **Events** represent something that “happens” during the process. Usually they have a **cause** (trigger) or an **impact** (result).
- 3 kinds of events :



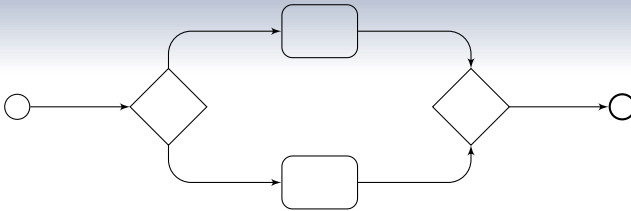
- Triggers and results :




- message catching (trigger) or throwing (result) ;
- time ;
- error catching or throwing ;
- signal catching or throwing ;
- complex condition (catching) ;
- ...



Flow objects : gateways

- Used to control the **divergence** and the **convergence** of the flow.



- Several kinds of gateways :
 - exclusive data-based  or event-based ;
 - inclusive  (condition based) ;
 - parallel  ;
 - ...



Connecting objects

- **Sequence flow** : used to show the order that activities will be performed in the process.



- **Message flow** : used to show a flow of messages between two entities (\neq pools).



- **Association** : used to associate a piece of information to a flow object (such as artifact, comment, input / output, ...).



or





Artifacts : data objects

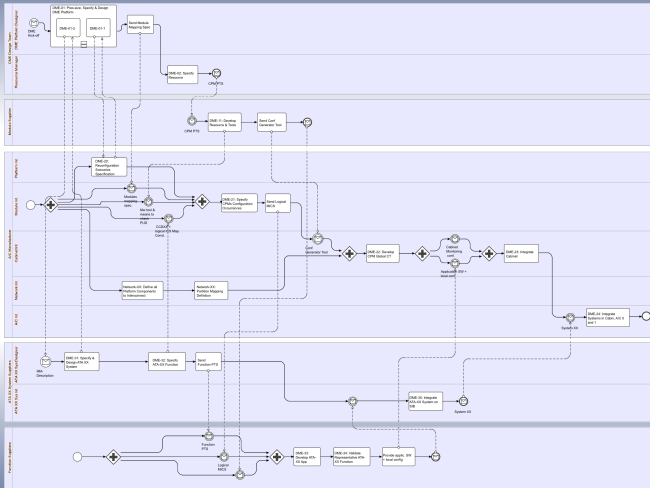
- Data objects are the only way of providing informations about :
 - what activities **need** to be performed (*i.e.* inputs) ;
 - what activities **produce** (*i.e.* outputs).

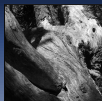


Logical MICS



The DME development process

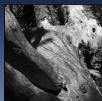




Editors and tools

A representative but not exhaustive selection of tools (see http://www.bpmn.org/BPMN_Supporters.htm that lists 53 active contributors to BPMN).

- **ILOG JViews** : a (non-deterministic) BPMN editor.
- **BxModeler** : an academic demonstrator of a collaborative online tool based on JSP and AJAX.
- **BizAgi** : a complete suite with execution, monitoring and editing tools. Very user-friendly, but not free software.
- **Intalio designer and server** : a graphical editor, and an execution environment through Web Services, based on the Eclipse framework.
- ...



Concluding remarks about BPMN

- A quite intuitive graphical language (easy to use and to understand).
- International standard.
- Plenty of existing tools.

However...

- Lack of formal semantics \rightsquigarrow ambiguities in the meaning of the diagrams.
- Poor support of data elements and activities inputs / outputs.



What's at the menu today?

- 1 **A short introduction to Business Process Modeling Notation**
 - BPMN: a snapshot
 - Core modeling elements: actors, flow objects, connectives and artifacts
 - Examples
 - Conclusion about BPMN
- 2 **Situation Calculus**
 - Situation Calculus in a Nutshell
 - The language of the situation calculus
 - Example
 - Extensions
- 3 **From BPMN to Situation Calculus**
 - Principles of the transcription
 - Examples
- 4 **Conclusion**



In the beginning was...

...An attempt to build an epistemically adequate language for **representing** and **reasoning about** the real world.

[MacCarthy and Hayes, 1969]

- Things evolve \rightsquigarrow **Situations**...
- ...Under the application of **Actions**.
- Properties that are true in a given situation can be false in another one \rightsquigarrow **Fluents**.
- We need a framework for representing and reasoning about facts \rightsquigarrow **First Order Logic**.



MacCarthy, J. and Hayes, P. (1969).

Some philosophical problems from the standpoint of artificial intelligence.

In Michie, D. and Melzer, B., editors, *Machine Intelligence 4*, pages 463–502. Edinburgh University Press.

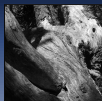


A glimpse at the language

Changes in the world result from **named actions**. A sequence of actions (a story) is called a **situation**.

- **Situations** are terms.
- S_0 : initial situation.
- **Actions** : functions (e.g. $put(x, y)$)
- $do(\alpha, s)$: the situation resulting from the execution of α in situation s .

Situation-dependant relations or functions are called **fluents**. For example : $workFor(x, y, s)$, $color(x, s)$.



Axiomatizing actions

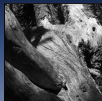
Axiomatizing actions usually mean describing their **preconditions** and **effects**.

- **Preconditions** describe the facts that must be true in order the action to be executable.

For example : $Poss(repair(r, x), s) \rightarrow hasGlue(r, s) \wedge broken(x, s)$.

- **Postconditions** or **effect axioms** describe the state of the world after the execution of an action.

For example : $fragile(x, s) \rightarrow broken(x, do(drop(r, x), s))$



Action qualification and frame problems

Two important reasoning problems arise :

- 1 **The qualification problem for actions** : What can we deduce from $Poss(repair(r, x), s) \rightarrow hasGlue(r, s) \wedge broken(x, s)$? Do these conditions characterize **all** the preconditions for the action to perform?
- 2 **The frame problem** :
 - Effect axioms are not enough. One must also provide axioms for things that **do not change** (e.g. $color(x, s) = c \rightarrow color(x, do(drop(r, y), s)) = c$)
 - Problem : $2 \times \mathcal{A} \times \mathcal{F}$ frame axioms! How to derive them automatically from effect axioms and how to express them in a succinct way?



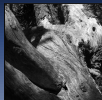
Reiter's proposal

① The qualification problem :

$$Poss(\text{repair}(r, x), s) \rightarrow \text{hasGlue}(r, s) \wedge \text{broken}(x, s)$$

② The frame problem :

$$\begin{aligned} \text{fragile}(x, s) &\rightarrow \text{broken}(x, \text{do}(\text{drop}(r, x), s)) \\ \text{nextTo}(b, x, s) &\rightarrow \text{broken}(x, \text{do}(\text{explode}(b), s)) \\ &\quad \neg \text{broken}(x, \text{do}(\text{repair}(r, x), s)) \end{aligned}$$



Reiter's proposal

① The qualification problem :

$$Poss(repair(r, x), s) \rightarrow hasGlue(r, s) \wedge broken(x, s)$$

Idealized approach ("minor" qualifications excluded)

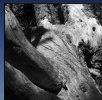
$$Poss(repair(r, x), s) \leftrightarrow hasGlue(r, s) \wedge broken(x, s)$$

② The frame problem :

$$fragile(x, s) \rightarrow broken(x, do(drop(r, x), s))$$

$$nextTo(b, x, s) \rightarrow broken(x, do(explode(b), s))$$

$$\neg broken(x, do(repair(r, x), s))$$



Reiter's proposal

① The qualification problem :

$$Poss(\text{repair}(r, x), s) \rightarrow \text{hasGlue}(r, s) \wedge \text{broken}(x, s)$$

Idealized approach ("minor" qualifications excluded)

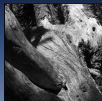
$$Poss(\text{repair}(r, x), s) \leftrightarrow \text{hasGlue}(r, s) \wedge \text{broken}(x, s)$$

② The frame problem :

$$\text{fragile}(x, s) \rightarrow \text{broken}(x, \text{do}(\text{drop}(r, x), s))$$

$$\text{nextTo}(b, x, s) \rightarrow \text{broken}(x, \text{do}(\text{explode}(b), s))$$

$$\neg \text{broken}(x, \text{do}(\text{repair}(r, x), s))$$



Reiter's proposal

1 The qualification problem :

$$Poss(\text{repair}(r, x), s) \rightarrow \text{hasGlue}(r, s) \wedge \text{broken}(x, s)$$

Idealized approach ("minor" qualifications excluded)

$$Poss(\text{repair}(r, x), s) \leftrightarrow \text{hasGlue}(r, s) \wedge \text{broken}(x, s)$$

2 The frame problem :

$$\text{fragile}(x, s) \rightarrow \text{broken}(x, \text{do}(\text{drop}(r, x), s))$$

$$\text{nextTo}(b, x, s) \rightarrow \text{broken}(x, \text{do}(\text{explode}(b), s))$$

$$\neg \text{broken}(x, \text{do}(\text{repair}(r, x), s))$$

Causal completeness axiom and quantification over actions.

$$\text{broken}(x, \text{do}(a, s)) \leftrightarrow (\exists r)a = \text{drop}(r, x) \wedge \text{fragile}(x, s)$$

$$\vee (\exists b)a = \text{explode}(b) \wedge \text{nextTo}(b, x, s)$$

$$\text{broken}(x, s) \wedge \neg(\exists r)a = \text{repair}(r, x)$$



Summary of Reiter's proposal

- Successor state axioms

$$F(\vec{x}, do(a, s)) \leftrightarrow \gamma_F^+(\vec{x}, a, s) \vee (F(\vec{x}, s) \wedge \neg \gamma_F^-(\vec{x}, a, s))$$

$$f(\vec{x}, do(a, s)) = y \leftrightarrow$$

$$\gamma_f(\vec{x}, y, a, s) \vee (f(\vec{x}, s) = y \wedge \neg \exists y' \gamma_f(\vec{x}, y', a, s))$$

- Precondition axioms

$$Poss(\alpha(\vec{x}), s) \leftrightarrow \Pi_\alpha(\vec{x}, s)$$



Reiter, R. (2001).

Knowledge in Action : Logical Foundations for Specifying and Implementing Dynamical Systems.

MIT Press.



The language of the Situation Calculus (i)

A second order many-sorted language with equality

- **Logical symbols** : $() \rightarrow \neg =$ and countably infinitely many variables $\{x, y, z, x_1, y_1, z_1, \dots\}$
- **Parameters (interpretation dependant symbols)** : \forall , predicates and functions.

Some other classical definitions :

- Terms, atoms, literals, well-formed formulas
- Free and bound variables.
- $\wedge, \vee, \leftrightarrow, \exists$.

Second-order constructions :

- **Predicate variables**
- **Function variables**



The language of the Situation Calculus (ii)

- Three sorts of variables : **action**, **situation**, and **object**.
- Two function symbols of sort **situation** : s_0 and $do : action \times action \rightarrow situation$.
- A binary predicate symbol \square : $situation \times situation$.
- A binary predicate symbol $Poss : action \times situation$.
- Situation independant predicates and functions (e.g. $height(mttoubkal)$)
- Action functions (e.g. $move(a, b)$).
- Relational and functional fluents (e.g. $ontable(a)$).



Foundational axioms

- We need among others, a **second order axiom**, representing the **induction** principle over situations. \leadsto Some intuitive reasoning tasks (such as planning) need second order inference mechanisms.
- We need an axiom to represent **executable** situations :

$$executable(s) \stackrel{def}{=} (\forall a, s^*) do(a, s^*) \sqsubseteq s \rightarrow Poss(a, s^*)$$



Some Hanoi(ing) example





Some Hanoi(ing) example

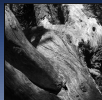




Some Hanoi(ing) example



- **Actions** : $move(D, R_1, R_2) \rightsquigarrow$ move disk D from rod R_1 to rod R_2 .
- **Fluents** :
 - $bottomRod(D, R, S) \rightsquigarrow$ in situation S , disk D lays in the first level of rod R ;
 - $topRod(D, R, S) \rightsquigarrow$ in situation S , disk D lays on the top of rod R ;
 - $on(R_1, R_2, S) \rightsquigarrow$ in situation S , disk R_1 is on disk R_2 ;
 - $empty(R, S) \rightsquigarrow$ in situation S , rod R is empty.
- **Situation independant facts** : $lt(d1, d2)$, $lt(d2, d3)$, $lt(d1, d3)$.



Hanoi again

- **Primitive action preconditions :**

$$\begin{aligned} \text{poss}(\text{move}(D, RX, RY), S) &\leftrightarrow \text{topRod}(D, RX, S) \\ &\quad \wedge (\text{empty}(RY, S) \vee \text{topRod}(DZ, RY, S) \wedge \text{lt}(D, DZ)) \end{aligned}$$

- **Successor state axioms :**

$$\begin{aligned} \text{bottomRod}(D, R, \text{do}(A, S)) &\leftrightarrow (\text{bottomRod}(D, R, S) \wedge A \neq \text{move}(D, R, R_2)) \\ &\quad \vee (A = \text{move}(D, R_3, R) \wedge \text{empty}(R, S)) \end{aligned}$$

$$\begin{aligned} \text{on}(DX, DY, \text{do}(A, S)) &\leftrightarrow \text{on}(DX, DY, S) \wedge A \neq \text{move}(DX, R_1, R_2) \\ &\quad \vee A = \text{move}(DX, R_3, R) \wedge \text{topRod}(DY, R, S) \end{aligned}$$

$$\begin{aligned} \text{topRod}(D, R, \text{do}(A, S)) &\leftrightarrow (\text{topRod}(D, R, S) \wedge A \neq \text{move}(D, R, R_2) \\ &\quad \wedge A \neq \text{move}(D_2, R_3, R)) \end{aligned}$$

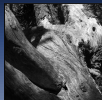
$$\vee A = \text{move}(D, R_4, R)$$

$$\vee A = \text{move}(DY, R, R_5) \wedge \text{on}(DY, D, S)$$

$$\text{empty}(R, \text{do}(A, S)) \leftrightarrow \text{empty}(R, S) \wedge A \neq \text{move}(D_2, R_2, R)$$

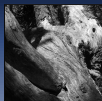
$$\vee A = \text{move}(D, R, R_3) \wedge \text{bottomRod}(D, R, S)$$

- **Initial situation :** $\text{bottomRod}(d3, r1, s0)$, $\text{on}(d2, d3, s0)$, $\text{on}(d1, d2, s0)$,
 $\text{topRod}(d1, r1, s0)$, $\text{empty}(r2, s0)$, $\text{empty}(r3, s0)$



Hanoi (queries)

- `?- poss(move(d2, r1, r3), do(move(d1, r1, r2), s0)).`
`true`
- `?- on(DX, DY, do(move(d1, r1, r3), s0)).`
`DX = d2,`
`DY = d3;`
- `?- top_rod(D, R, do(move(d1, r1, r2), s0)).`
`D = d1,`
`R = r2;`
`D = d2,`
`R = r1;`



Extensions

- **Golog** :

① **Primitive actions** : $Do(a, s, s') \stackrel{def}{=} (Poss(a[s], s) \wedge s' = do(a[s], s))$,

② **Test actions** : $Do(\varphi?, s, s') \stackrel{def}{=} (\varphi[s] \wedge s = s')$,

③ **Sequence** : $Do([\delta_1; \delta_2], s, s') \stackrel{def}{=} \exists s^*. (Do(\delta_1, s, s^*) \wedge Do(\delta_2, s^*, s'))$.

④ **Non deterministic choice of actions** :

$Do((\delta_1 | \delta_2), s, s') \stackrel{def}{=} Do(\delta_1, s, s') \vee Do(\delta_2, s, s')$.

⑤ **Non deterministic choice of arguments** :

$Do((\pi x)\delta(x), s, s') \stackrel{def}{=} \exists x. Do(\delta(x), s, s')$.

⑥ **Non deterministic iteration** :

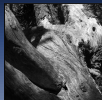
$Do(\delta^*, s, s') \stackrel{def}{=} \forall P. \{ \forall s_1 P(s_1, s_1) \wedge \forall s_1, s_2, s_3 [P(s_1, s_2) \wedge Do(\delta, s_2, s_3) \rightarrow P(s_1, s_3)] \} \rightarrow P(s, s')$.

⑦ **Procedures** (as macros)

⑧ if-then-else and while loops.

- **Congolog** : Time, concurrency and processes.

- **RGolog** : Exogenous actions, interrupts.



Planning in Situation Calculus

Usual planners can be programmed in Golog (e.g. breadth-first search planner).

Example (Back to Hanoi) :

```
?- planbf(10).
```

```
Success. Good situations : 1385
```

```
[move(d1, r1, r2), move(d2, r1, r3), move(d1, r2, r3),  
move(d3, r1, r2), move(d1, r3, r1), move(d2, r3, r2),  
move(d1, r1, r2)]
```

```
More? y.
```



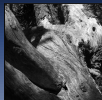
What's at the menu today?

- 1 **A short introduction to Business Process Modeling Notation**
 - BPMN: a snapshot
 - Core modeling elements: actors, flow objects, connectives and artifacts
 - Examples
 - Conclusion about BPMN
- 2 **Situation Calculus**
 - Situation Calculus in a Nutshell
 - The language of the situation calculus
 - Example
 - Extensions
- 3 **From BPMN to Situation Calculus**
 - Principles of the transcription
 - Examples
- 4 **Conclusion**



Basic transcription rules

- **Pools and lanes** : actor and subActor
- **Activities** : two actions $startTask(x, y)$ and $endTask(x, y)$ and one fluent $currentTask(x, y)$, even for compound activities.
- **Connexion flow elements** : $currentFlow(f_o, f_o', s)$.
- **Events** : events are exogenous or endogenous actions (**e.g.** $msg(a_1, a_2)$).
- **Gateways** : action $pass(f_g, y)$, whose preconditions and effects depend on the nature of the gateway (splitting, merging), and on its type (parallel, exclusive, inclusive, ...).
- **Data artifacts** : fluents $status(F_\alpha, S)$.



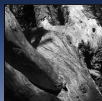
Atomic tasks : Preconditions and SSA

- Action precondition axioms :

$$\begin{aligned} \text{poss}(\text{startTask}(f_t, Y), S) &\leftrightarrow \text{currentFlow}(f_{\text{pred}(t)}, f_t, S) \wedge \neg \text{currentTask}(f_t, Y, S) \\ \text{poss}(\text{endTask}(f_t, Y), S) &\leftrightarrow \text{currentTask}(f_t, Y, S) \end{aligned}$$

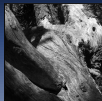
- Successor state axioms :

$$\begin{aligned} \text{currentTask}(f_t, Y, \text{do}(A, S)) &\leftrightarrow A = \text{startTask}(X, Y) \\ &\quad \vee (\text{currentTask}(X, S) \wedge A \neq \text{endTask}(X, Y)) \\ \text{currentFlow}(X, Y, \text{do}(A, S)) &\leftrightarrow (\exists Z. A = \text{endTask}(X, Z) \wedge Y = \text{succ}(x)) \\ &\quad \vee (\text{currentFlow}(X, Y, S) \wedge a \neq \text{startTask}(X, Y)) \end{aligned}$$

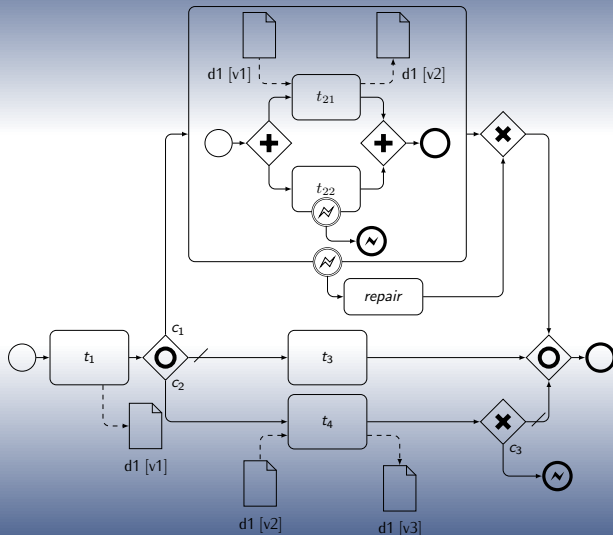


Features

- Able to model non instantaneous activities with instantaneous actions.
- \rightsquigarrow concurrency of activities appears naturally (same idea as in concurrent situation calculus).
- First order logic is powerful enough to express intuitively most natural facts about the world.
- Theoretically, the translation from BPMN to Situation Calculus can be made automatically.
- Computational efficiency?



Back to the toy example





The toy example : model and queries

Time for demonstration.



The DME development process

- The DME development process has been translated from BPMN to a theory of action in the situation calculus.
- Although some particular queries can be answered efficiently by the Prolog interpreter, most usual queries (such that : finding a plan) seem to be untractable.



What's at the menu today?

- 1 **A short introduction to Business Process Modeling Notation**
 - BPMN: a snapshot
 - Core modeling elements: actors, flow objects, connectives and artifacts
 - Examples
 - Conclusion about BPMN
- 2 **Situation Calculus**
 - Situation Calculus in a Nutshell
 - The language of the situation calculus
 - Example
 - Extensions
- 3 **From BPMN to Situation Calculus**
 - Principles of the transcription
 - Examples
- 4 **Conclusion**



What can we conclude ?

On the one hand :

- BPMN seems to be an intuitive modeling language.
- Situation calculus provide a powerful yet intuitive way to reason about processes.
- Using some reasonable assumptions, automatic translations from BPMN diagrams to theories of actions are theoretically possible.

On the other hand :

- Is it really revolutionary ?
- Is it really practically usable (computational efficiency) ?
- What's next ?
- Can't we use anything else to reason more efficiently (e.g. Petri nets) ?