Business Process Management

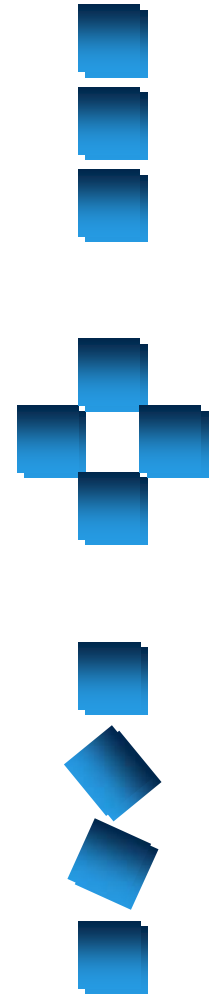
Elective in Software and Services

Massimiliano de Leoni
DIS – SAPIENZA, Università di Roma
deleoni@dis.uniroma1.it

www.dis.uniroma1.it/~deleoni

Acknowledgement

- This presentation uses some slides prepared by the following people:
  - Wil van der Aalst, TUE & QUT
  - Michael Adams, QUT
  - Lachlan Aldred, QUT
  - Arthur ter Hofstede, QUT
  - Marcello La Rosa, QUT
  - Nick Russell, QUT
  - Petia Wohed, SU/KTH
  - Moe Wynn, QUT



Terminology

- Workflow
  - "The automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action according to a set of procedural rules."
    WfMC, Terminology & Glossary, WFMC-TC-1011 3.0, February 1999
- Workflow Management System (WfMS)
  - a.k.a. Business Process Management Systems (BPMS)
  - "A system that completely defines, manages and executes workflows through the execution of software whose order of execution is driven by a computer representation of the workflow logic."
    WfMC, Terminology & Glossary, WFMC-TC-1011 3.0, February 1999
- An alternative definition:
  - "A software system that manages and executes operational processes involving people, applications, and/or resources on the bases of process models."
    M. Dumas, W. van der Aalst, A. ter Hofstede, Process-Aware Information Systems: Bridging People and Software through Process Technology, John Wiley & Sons, 2005

Background

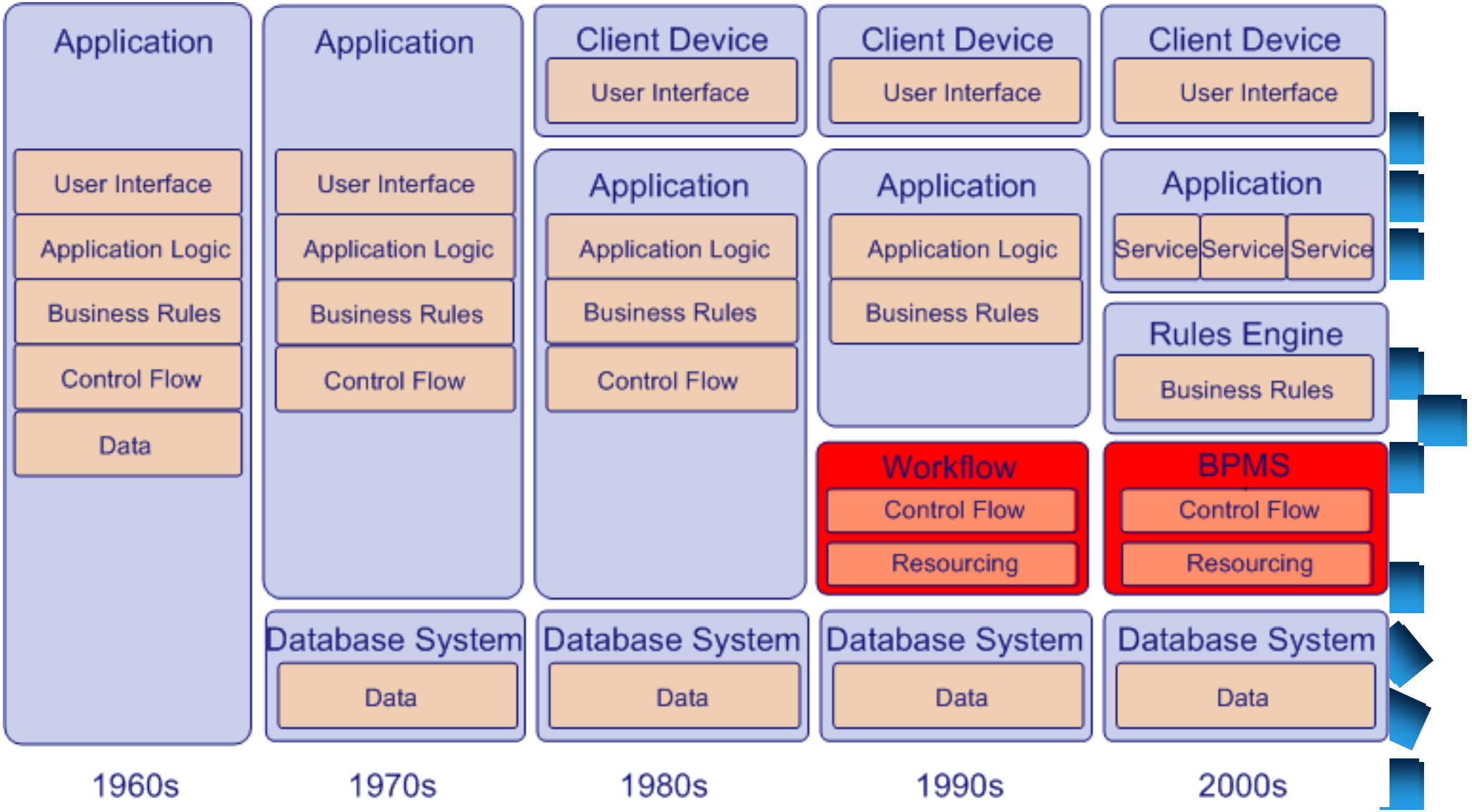
- Business Process Management (BPM) perceived internationally as top business priority
  - BPM views processes as central in an organization
- Significant business benefits can be derived from its (correct) application
  - Potential for substantial cost & time savings
- The field touches upon both business and IT
  - Managerial and technical issues are taken into account
- Business Process Automation, essentially **Workflow Management** through BPMS offers a number of benefits
  - Business Process Automation, is a subfield of BPM

Pros of Workflow Management

- Process models should serve as the blueprint for subsequent automated support

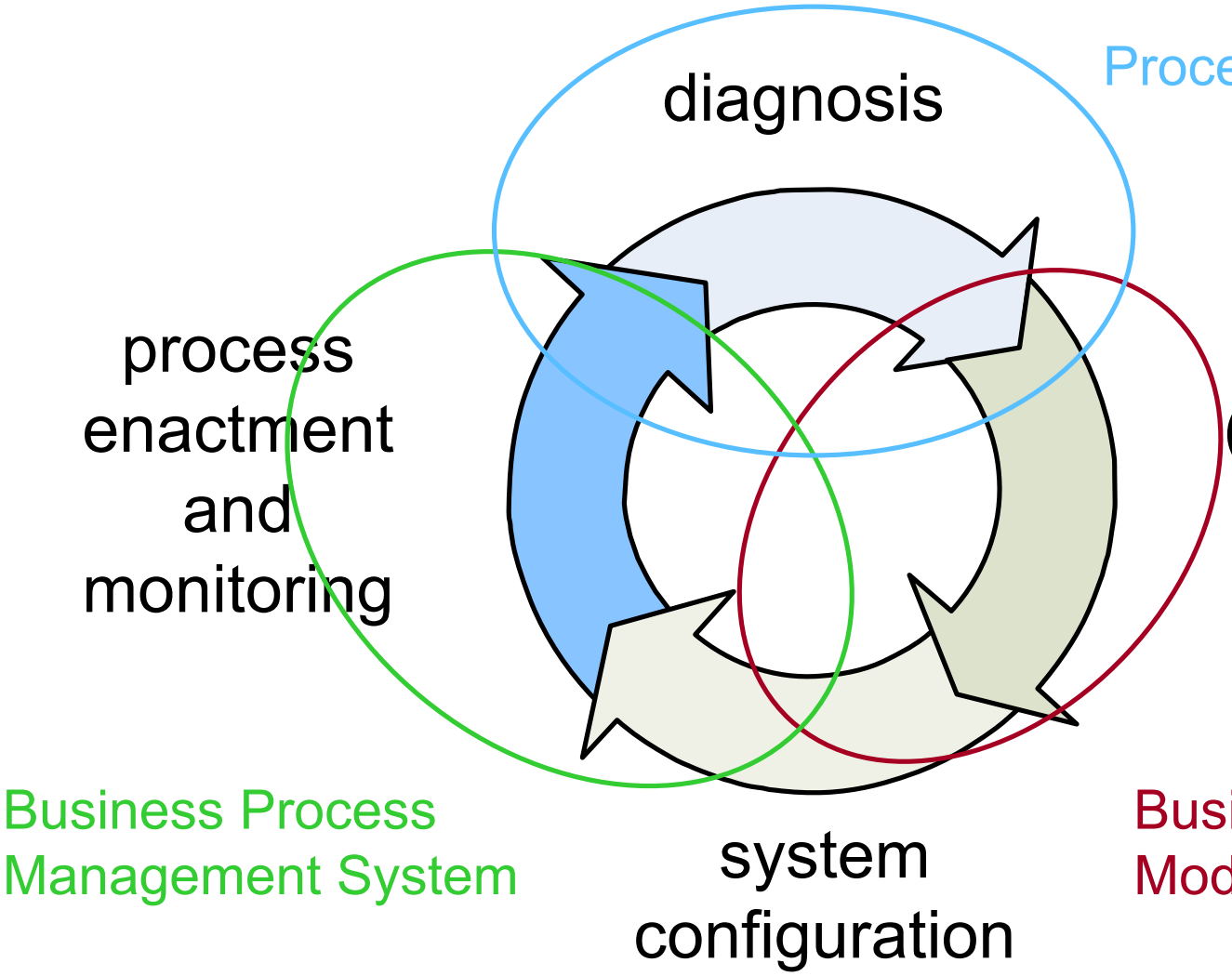
- Explicit representation of control-flow
  - Process model changes do not require low-level coding efforts
- Explicit representation of resource involvement
  - Work can directly be routed to the right resources
  - Aspects such as workload and work history can be taken into account in work assignment
- Coupling of processes and data assists with data accuracy
- Monitoring support
  - Identification and resolution of bottlenecks
- Post-execution analysis (Process Mining)
  - Identification of opportunities for process improvement

BPM Evolution



(Source: Adapted from Chapter 2 of AHM ter Hofstede et al (editors), Modern Business Process Automation: YAWL and its Support Environment, Springer 2009; Also used in Nick Russell et al "Workflow Patterns: The Definitive Guide" (to appear); based on figure by Wil van der Aalst)

BPM Life-Cycle



A. ter Hofstede, W. van der Aalst, M. Adams, N. Russell, *Modern Business Process Automation: YAWL and its Support Environment*, Springer, 2009

Process Specifications need to model different perspectives

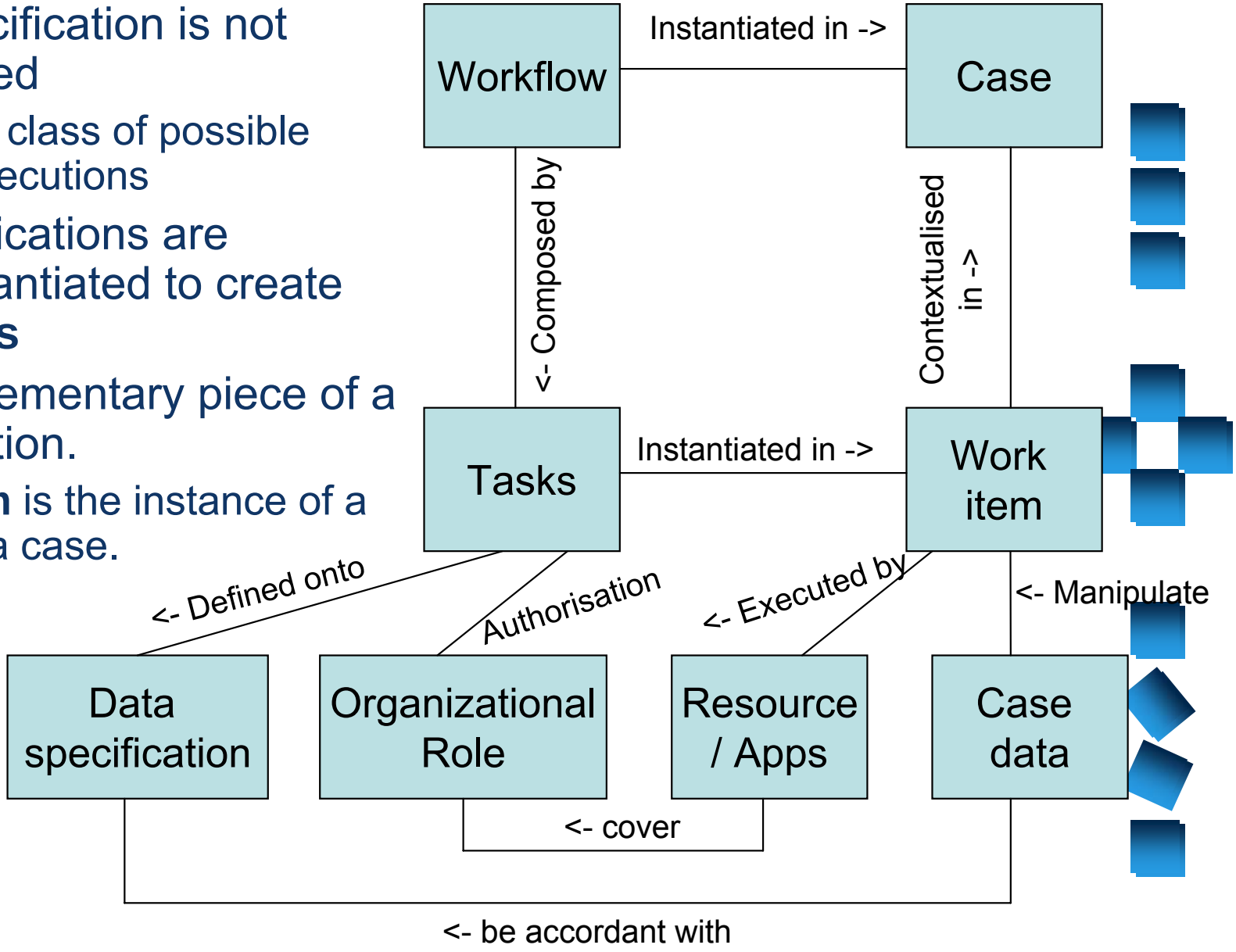
- Control-Flow
  - Which tasks need to be executed and in what order
- Data
  - What data elements exist, to whom are they visible, how are they passed on
- Resources
  - Who is authorised to execute certain tasks, are tasked assigned by the system or can participants volunteer for their execution, on what basis is work assigned

Sometimes these perspectives are explained in terms of *Who (Resource), What (Data)* and *When (Control-flow)*

These perspectives follow S. Jablonski and C. Bussler's classification from:
Workflow Management: Modeling Concepts, Architecture, and Implementation. International Thomson Computer Press, 1996

The workflow concepts

- A process specification is not directly executed
  - It denotes a class of possible workflow executions
- Process specifications are concretely instantiated to create so-called **cases**
- A task is the elementary piece of a process execution.
  - A **work item** is the instance of a task inside a case.



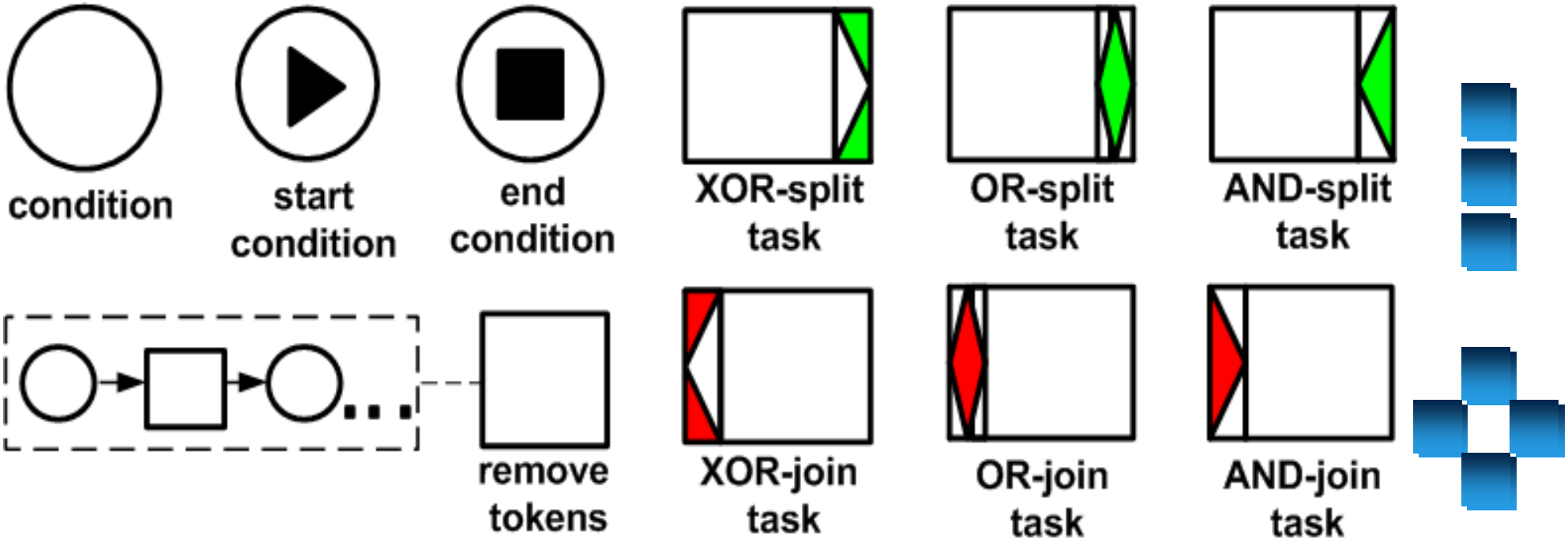
Process modelling

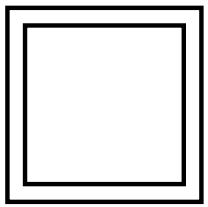
- No consensus has been reached to describe executable processes
- Several alternatives have been proposed, but none has become a standard commonly recognised.
  - Lack of commonly accepted conceptual foundations
- Some process modelling languages are based on a formal unambiguous semantics that can be input for BPMS
  - Explicit representation of control flow dependencies and resourcing strategies
  - A formal background is required by process designers
- Some languages are high level and intended for non-expert users.
  - They came with nice graphical representations that are vague but useful for an initial insight.

Emergence of YAWL

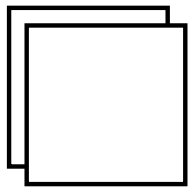
- Defined by Wil van der Aalst and Arthur ter Hofstede in 2002
- Yet Another Workflow Language
- Intention: to provide comprehensive support for the workflow patterns
- Inspired by Workflow nets, but with direct support for
  - Cancelation
  - Multiple executions of the same task in the same process instance
  - Synchronization of active paths only (OR-join)
- Formal semantics

YAWL notation



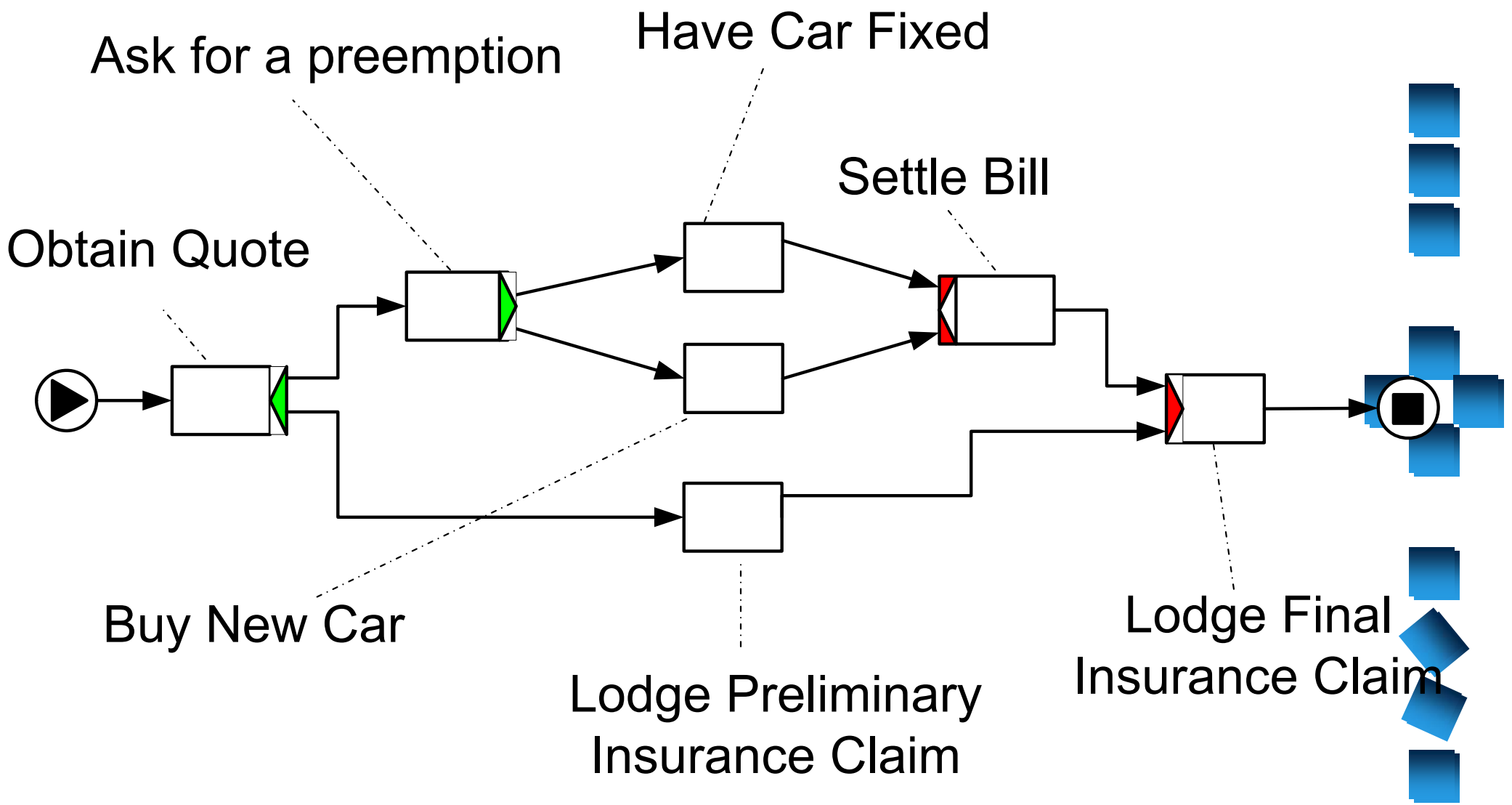


Composite task



Multiple Instance task

YAWL: A Simple Example



The four components of a BPM Suite

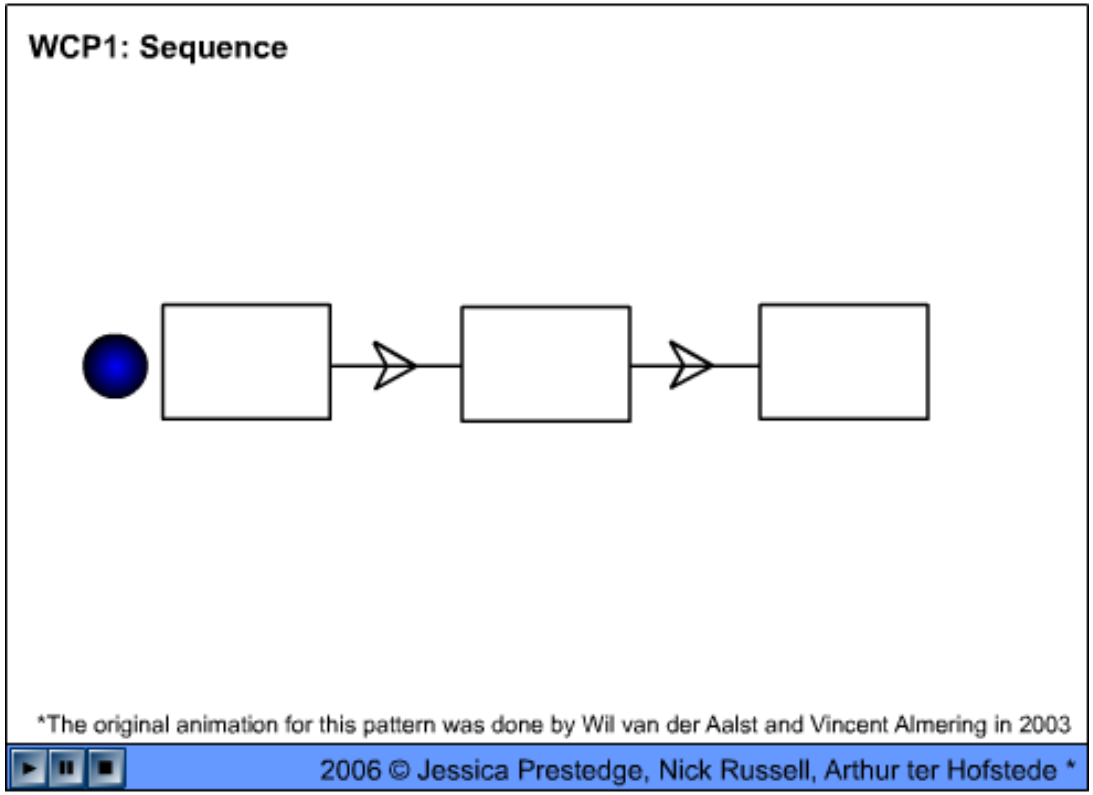
- A modelling tool
  - It provides Graphic User Interface to design process specifications to be given as input to a BPMS Engine
- A Business Process Management System Engine
  - It stores and interprets process specifications, creates and manages cases as they are instantiated, and controls their interaction with workflow participants and applications.
  - Typically, BPMS offers work items to all resources that qualify
    - The first resource that selects a work item is the only executing it.
- A work-list handler
  - It allows administrators to create, manage and monitor cases
  - Process participants may access to the queues of work items assigned, running, etc, letting them continuing with their execution
- Analysis tools
  - They are intended to mine the log of past executions to find recurrent patterns, deviations from the expected behaviour, etc

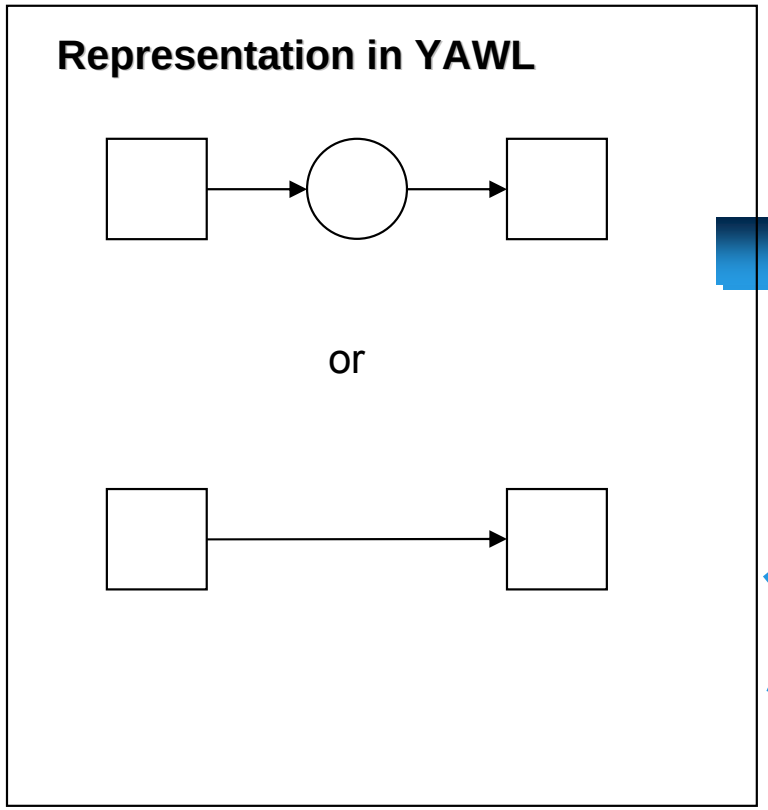
Workflow Patterns Initiative

- Started in 1999, joint work TU/e and QUT
- Objectives:
  - Identification of workflow modelling scenarios and solutions
  - Benchmarking
    - Workflow products (MQ/Series Workflow, Staffware, etc)
    - Proposed standards for web service composition (BPML, BPEL)
    - Process modelling languages (UML, BPMN)
    - Open Source BPM offerings (jBPM, OpenWFE, Enhydra Shark)
  - Foundation for selecting workflow solutions
- Home Page: **www.workflowpatterns.com**
  - Animations of the patterns are available on the web site
- Primary publication:
  - W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, A.P. Barros, “Workflow Patterns”, Distributed and Parallel Databases 14(3):5-51, 2003.
- Evaluations of commercial offerings, research prototypes, proposed standards for web service composition, etc

Sequence

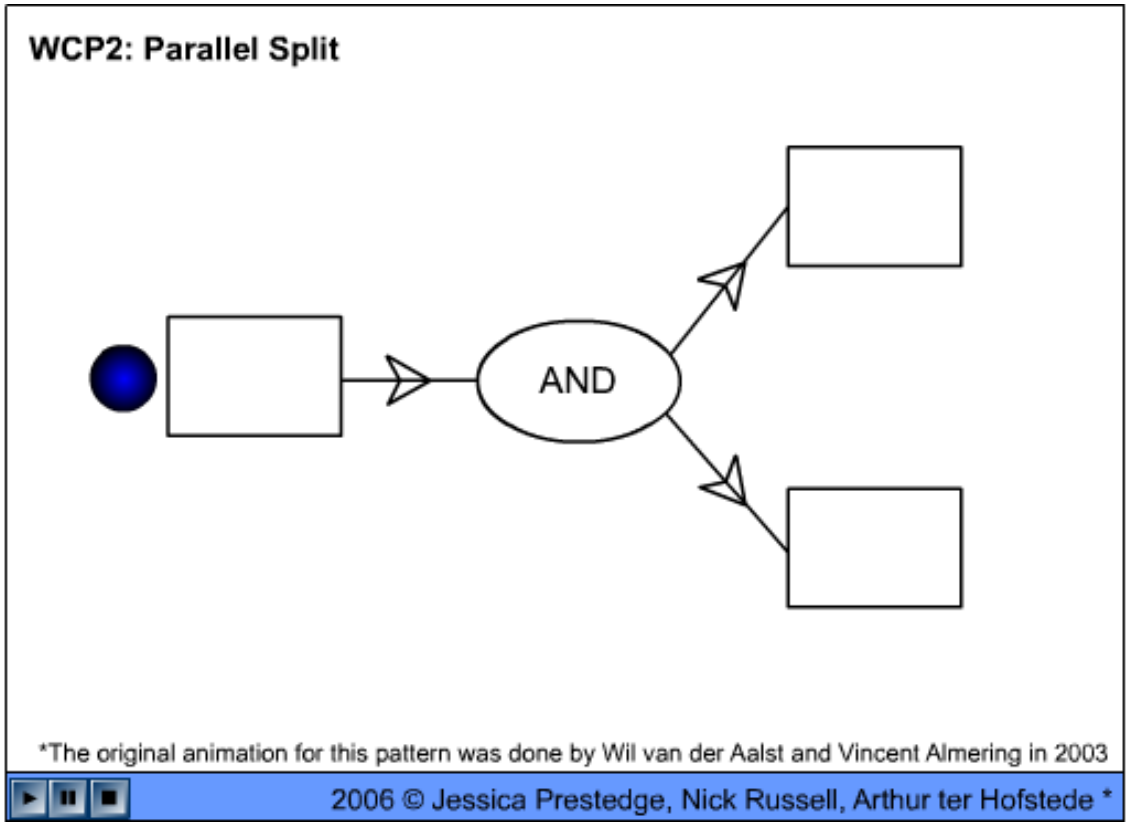
An activity in a workflow process is enabled after the completion of a preceding activity in the same process.

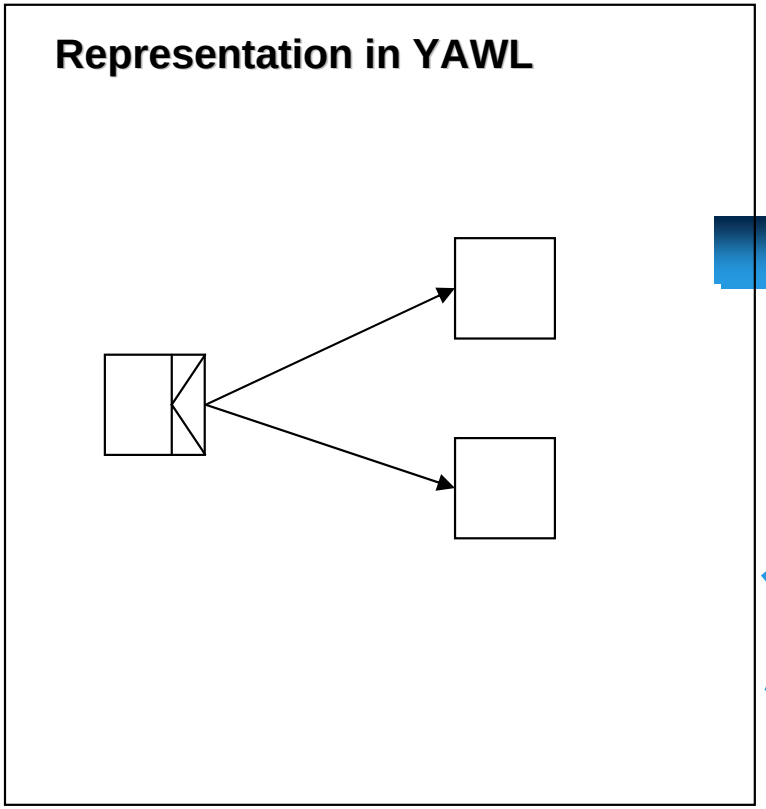




Parallel Split (AND-split)

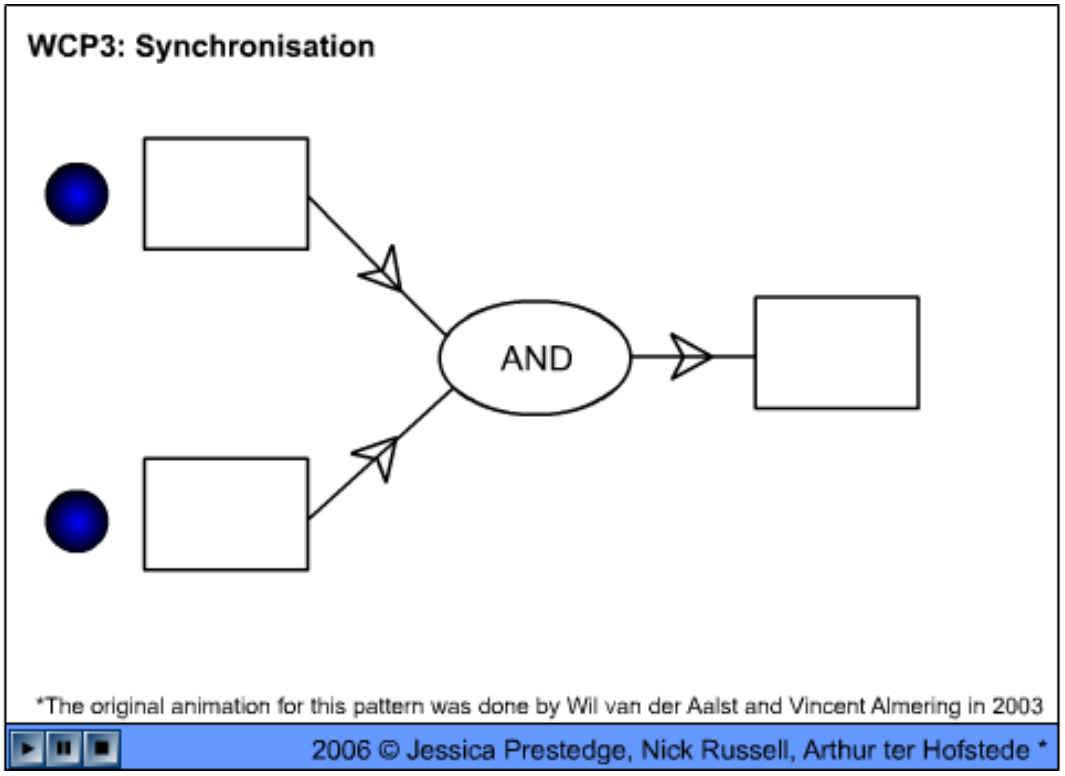
The divergence of a branch into two or more parallel branches each of which execute concurrently.

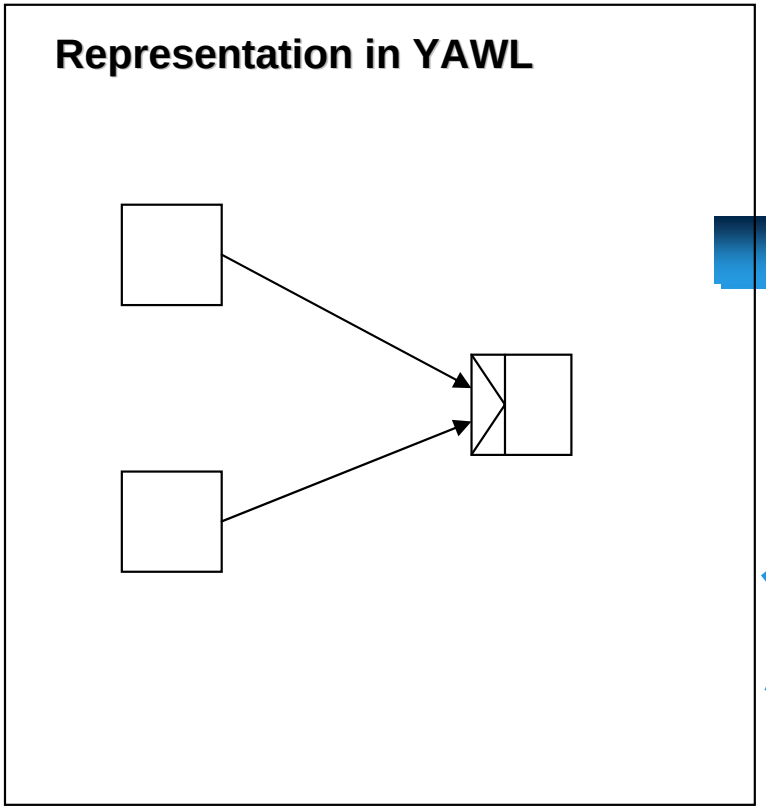




Synchronisation (AND-join)

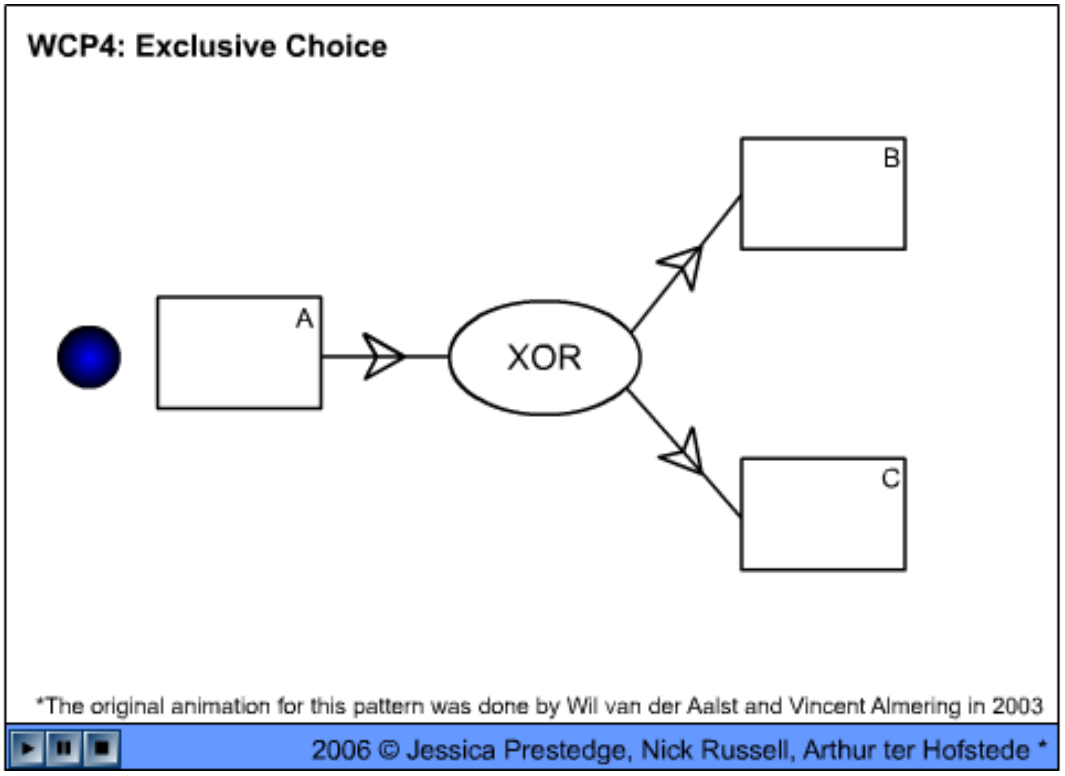
The convergence of two or more branches into a single subsequent branch such that the thread of control is passed to the subsequent branch when all input branches have been enabled.

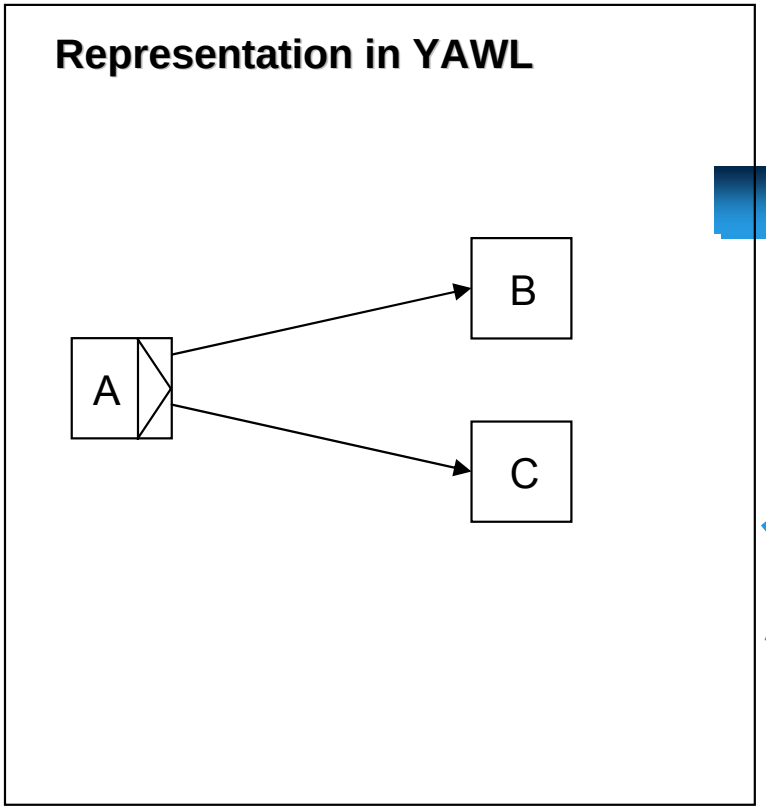




Exclusive Choice (XOR-split)

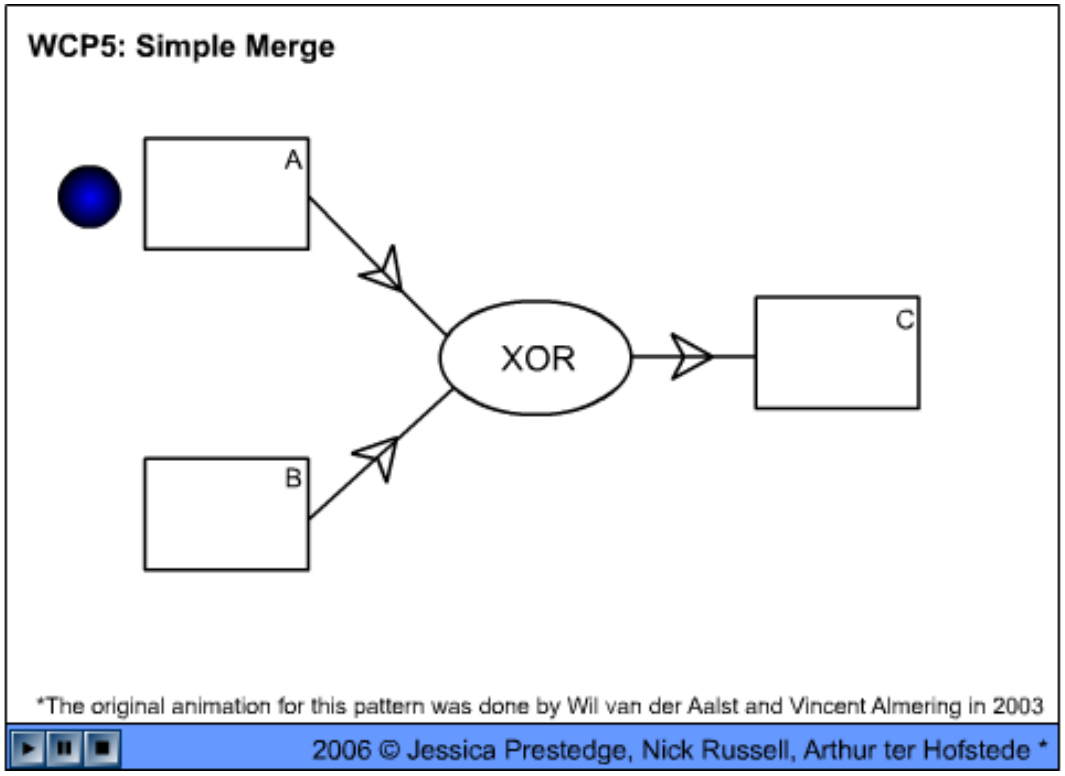
The divergence of a branch into two or more branches such that when the incoming branch is enabled, the thread of control is immediately passed to precisely one of the outgoing branches based on a mechanism that can select one of the outgoing branches

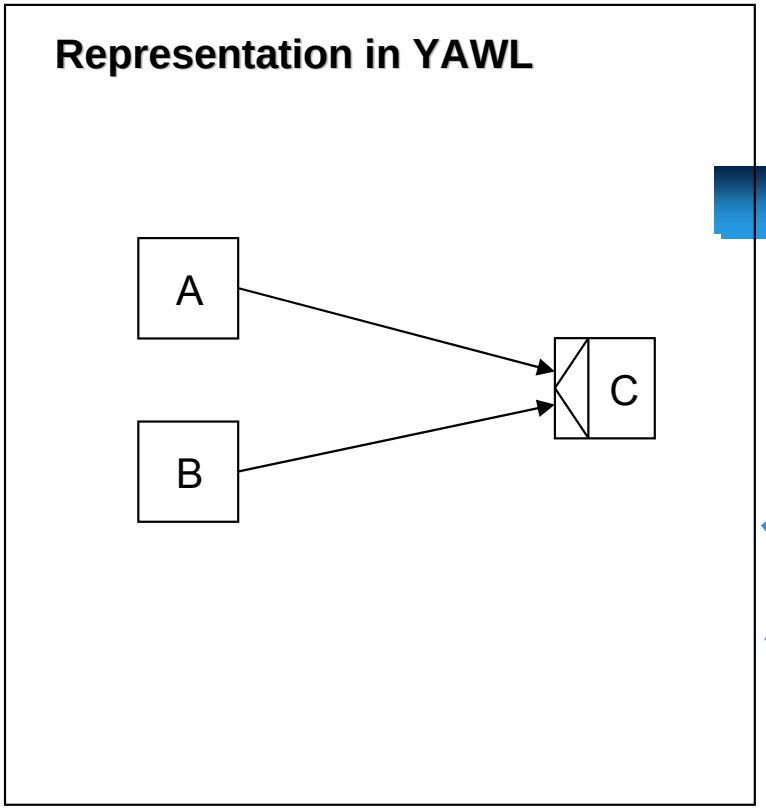




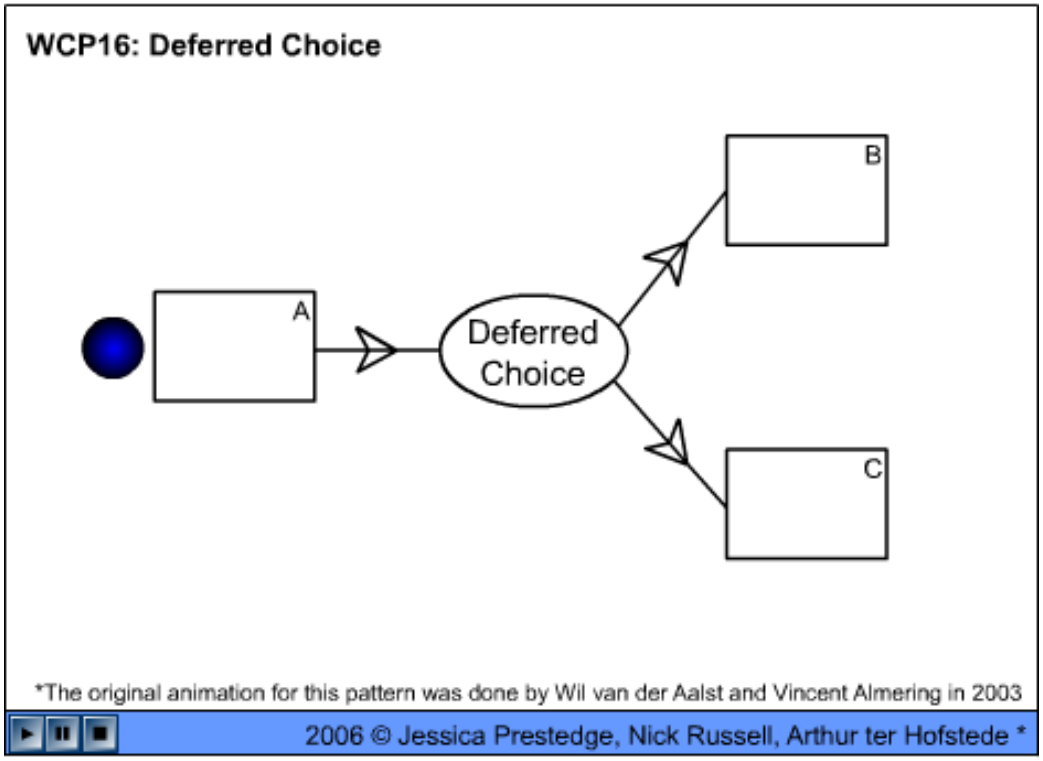
Simple Merge (XOR-join)

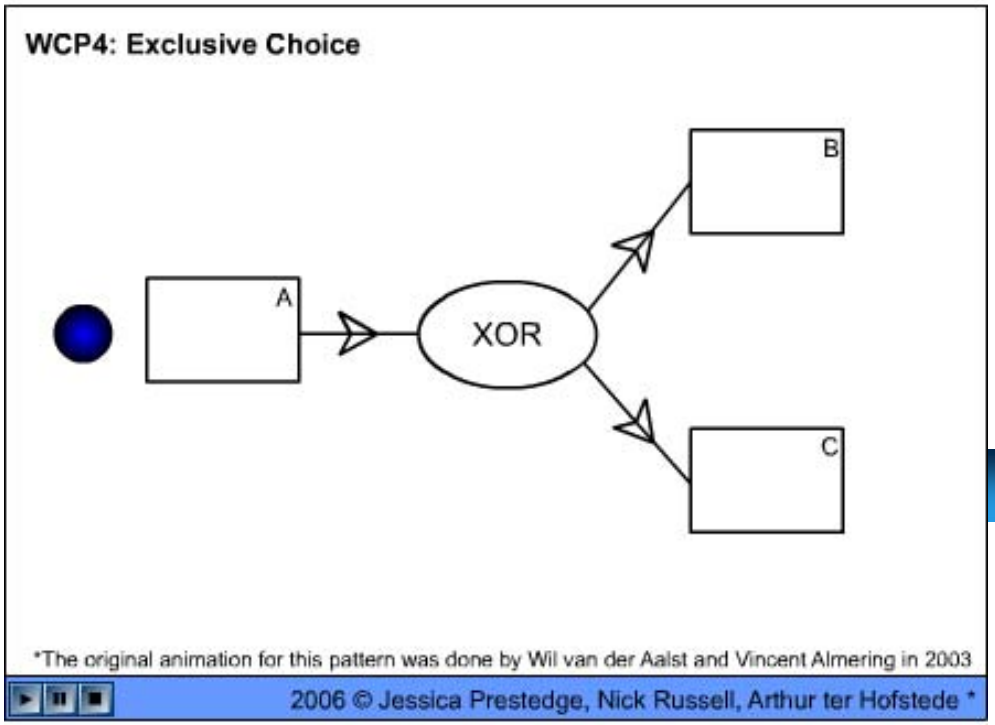
The convergence of two or more branches into a single subsequent branch such that each enablement of an incoming branch results in the thread of control being passed to the subsequent branch.





Deferred Choice vs Exclusive Choice

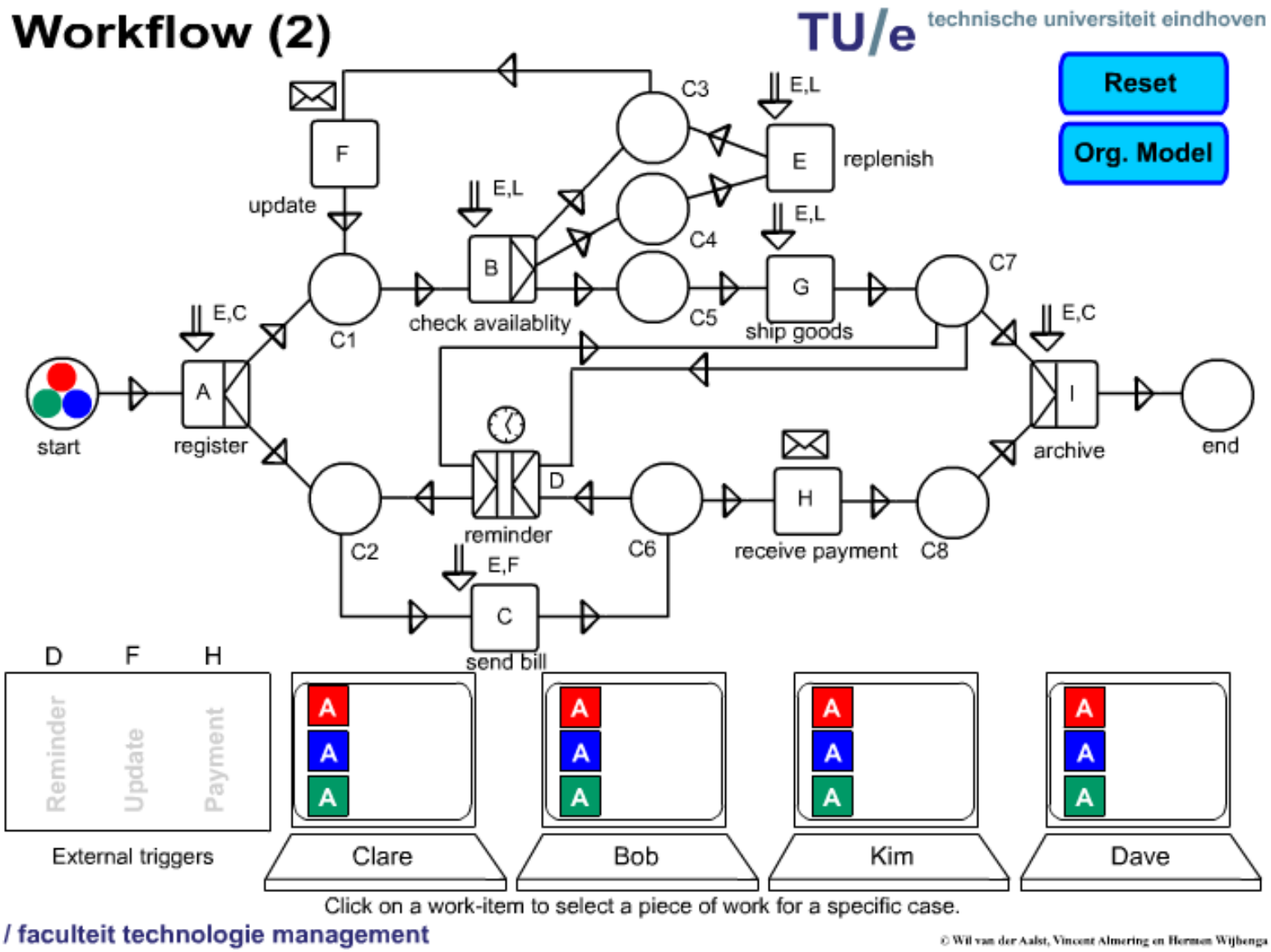




- Choice made by the environment not the system
  - E.g., after receiving some external events (such as a mail delivery, a time expiration

- Choice made by the system on the basis of case data

Workflow Animation



© Wil van der Aalst, Vincent Almering and Herman Wijbenga

Extended Workflow Net

- A workflow process definition in YAWL is based on the definition of *extended workflow net*

An *extended workflow net (EWF-net)* is a tuple (**C, i, o, T, F, *split, join, rem, nofi***) such that

- **C** is a set of conditions
- **i** $\in$ **C, o** $\in$ **C** denote, respectively, the initial and final condition
- **T** is a set of tasks, where **C** $\cap$ **T =** $\varnothing$
- **F** $\subseteq$ **(C – {o} x T)** $\cup$ **(T x C – {i})** $\cup$ **(T x T)** is a flow relation, s.t. every node **C** $\cap$ **T** is on a directed path from **i** to **o**.
- ***split:*** **T** $\nrightarrow$ **{***And, Xor, Or***}** is a partial map assigning a split behaviour to a task
- ***join:*** **T** $\nrightarrow$ **{***And, Xor, Or***}** is a partial map assigning a join behaviour to a task
- ***rem:*** **T** $\nrightarrow$ $\mathcal{P}$**(T** $\cup$ **C – {i,o})** defines parts of the net that need to be cleaned from tokens when task **T** is executed. $\mathcal{P}$**(S)** denotes the power set of **S**.
- ***nofi:*** **T** $\nrightarrow \mathcal{N}$ **x** $\mathcal{N}^{\infty}$ **x** $\mathcal{N}^{\infty}$ **x {***dynamic, static***}** denotes a partial function that specifies the number of instances of each task (min, max, threshold for continuation, dynamic/static instances creation). $\mathcal{N}^{\infty}$ indicates the set of natural numbers plus the infinite value symbol

W.M.P. van der Aalst and A.H.M. ter Hofstede. YAWL: Yet Another Workflow Language, Information Systems 30(4):245-275, 2005

A YAWL workflow specification

- A *YAWL workflow specification S* is a tuple **(Q,*top*,$\mathcal{T}$,*map*)** such that
  - **Q** is a set of extended workflow nets
  - ***top*** $\in$ **Q** is the top level workflow net
  - $\mathcal{T} = \bigcup_{n \in Q} T_n$ is the set of all tasks.
    - Tasks and Conditions are disjoint among all extended workflow nets: $(C_{N1} \cup T_{N1}) \cap (C_{N2} \cup T_{N1}) = \emptyset, \forall$ N1, N2 $\in$ Q
  - ***map***: $\mathcal{T} \nrightarrow$ **Q – {*top*}** is a function that maps each composite task onto an extended workflow net
- Hence, each task t $\in \mathcal{T}$ s.t. map(t) is defined is a composite task.
- For all extended workflow net N $\in$ Q - {top}:
  - $\exists$ t $\in \mathcal{T}$ s.t. map(t) = N
  - If $\exists$ $t_1, t_2 \in \mathcal{T}$, $\exists$ N $\in$ Q s.t. map(t1) = N $\wedge$ map(t2) = N $\Rightarrow$ $t_1 = t_2$

Advanced topics: Structural Verification

- Verification is concerned with determining, **in advance**, whether a process model exhibits certain desirable behaviours.
- **Option to complete**
  - A process when started can always complete
  - A process when started can complete in some situations (**Weak** option to complete)
- **Proper completion**
  - it should not have any other tasks still running for that process when the process ends
- **No dead tasks**
  - the process should not contain tasks that will never be executed

Explicit Extended Workflow Net

- YAWL is provided with a shorthand notation such that placing conditions are often unnecessary
- For the problem the structural verification, every condition should be explicitly defined

- Explicit EWF-net (E2WF-net) is a new EWF-net obtained from an initial one by adding conditions between tasks with direct connection
- Let N = (**C, i, o, T, F, *split, join, rem, nofi***) be an EWFnet, the corresponding **E2WF-net** is defined as (**C^{ext}, i, o, T, F^{ext}, *split, join, rem, nofi***) where
  - $C^{ext} = C \cup \{c_{(t1,t2)} \mid (t_1, t_2) \in F \cap (T \times T)\}$
  - $F^{ext} = (F \setminus (T \times T)) \cup \{ (t_1, c_{(t1,t2)}) \mid (t_1, t_2) \in F \cap (T \times T)\}$ $\cup \{ (c_{(t1,t2)}, t_2) \mid (t_1, t_2) \in F \cap (T \times T)\}$

Moe Thandar Wynn, David Edmond, Wil M. P. van der Aalst, Arthur H. M. ter Hofstede: Achieving a General, Formal and Decidable Approach to the OR-Join in Workflow Using Reset Nets. Proceedings of the Applications and Theory of Petri Nets (2005), pages 423-443

Some additional definitions

- Let N be an E2WF-net and $x \in (C^{ext} \cup T)$, we denote with $\bullet x$ and $x \bullet$ to denote the set of inputs and outputs of a node:
  - $\bullet x = \{y \mid (y,x) \in F^{ext}\}$
  - $x \bullet = \{y \mid (x,y) \in F^{ext}\}$
- The state of a E2WF-net can be represented as a **marking function M:C→$\mathcal{N}$** where *M(c)* returns the number of tokens in a condition *c*.
- Tasks are the active components of an E2WF-net and when a task *t* fires at a marking M, it changes to a new state, reaching a new marking M'.
  - It is denoted as: $M \rightarrow^{t} M'$
  - A sequence of arbitrary firing tasks can be denoted as: $M \rightarrow^{*} M''$
- Give two marking functions M_a, M_b defined on the same E2WF-net, we denote the following:
  - M_o is s.t. $\forall c \in C - \{o\}\ M_o(c) = 0$ and $M_o(\mathbf{o})=1$
  - M_i is s.t. $\forall c \in C - \{i\}\ M_o(c) = 0$ and $M_o(\mathbf{i})=1$
  - $M_a > M_b$ if $\forall c \in C\ M_a(c) > M_b(c)$
  - $M_a = M_b$ if $\forall c \in C\ M_a(c) = M_b(c)$

A formal definition of soundness

- Let N be an E2WF-net and M_i,M_o be the initial and final marking function, N is sound if and only if:
  1. *Option to complete*: for every marking M reachable from M_i, there exists an occurrence sequence leading from M to M_o:

$$\forall\ M\ (M_i \rightarrow^* M) \Rightarrow (M \rightarrow^* M_o)$$

  2. *Proper completion*: the marking M_o is the only marking reachable from M_i with at least one token in condition ***o***:

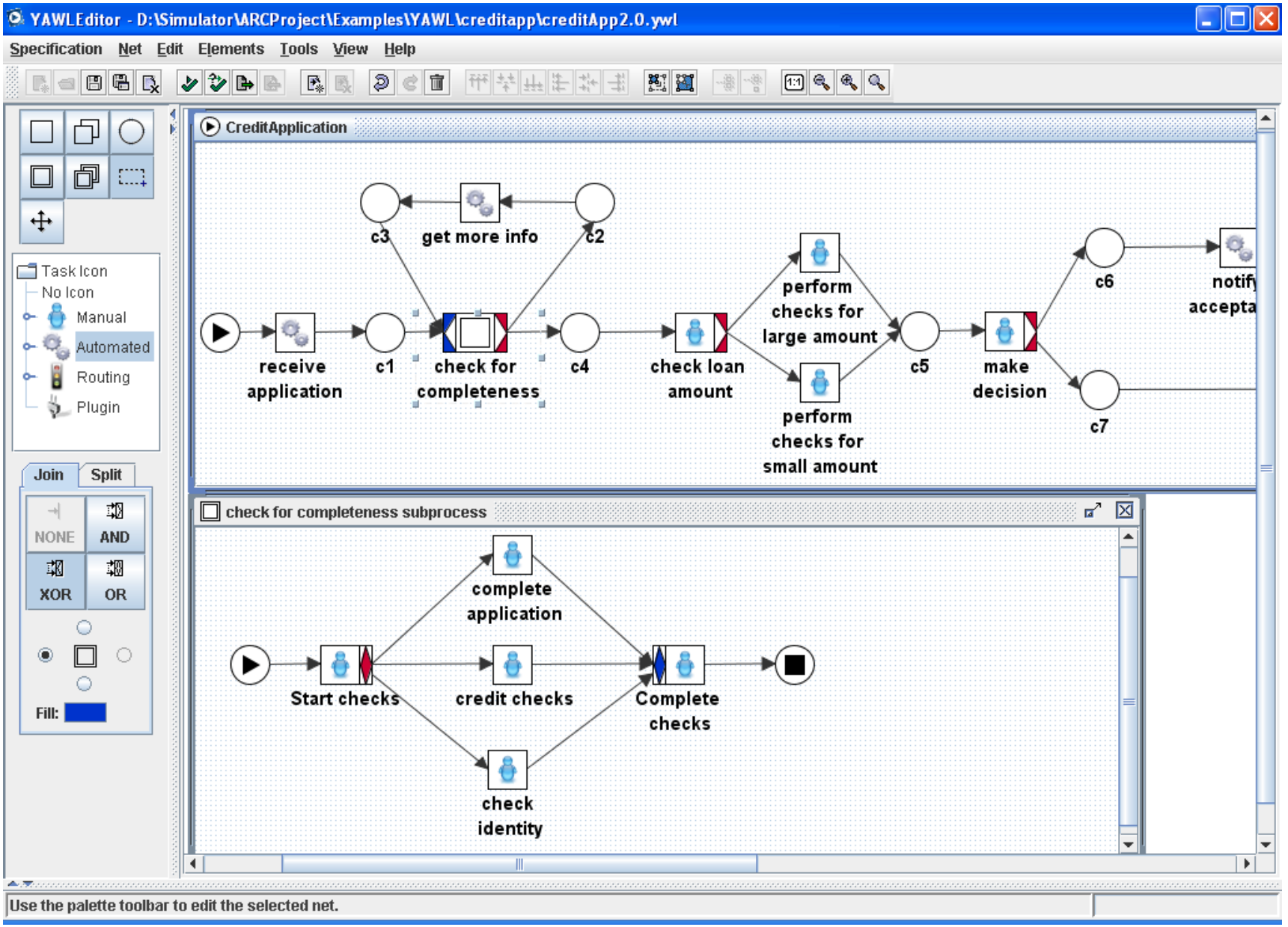
$$\forall\ M\ (M_i \rightarrow^* M \wedge M > M_o\ ) \Rightarrow (M = M_o)$$

     - When the final condition *o* is reached, no token is still presented in the other conditions.
  3. *No dead task*: for every task $t \in T$, there is a marking M reachable from M_i such that t is enabled:

$$\forall\ t \in T,\ \exists M,M'\colon M_i \rightarrow^* M \rightarrow^t M'$$

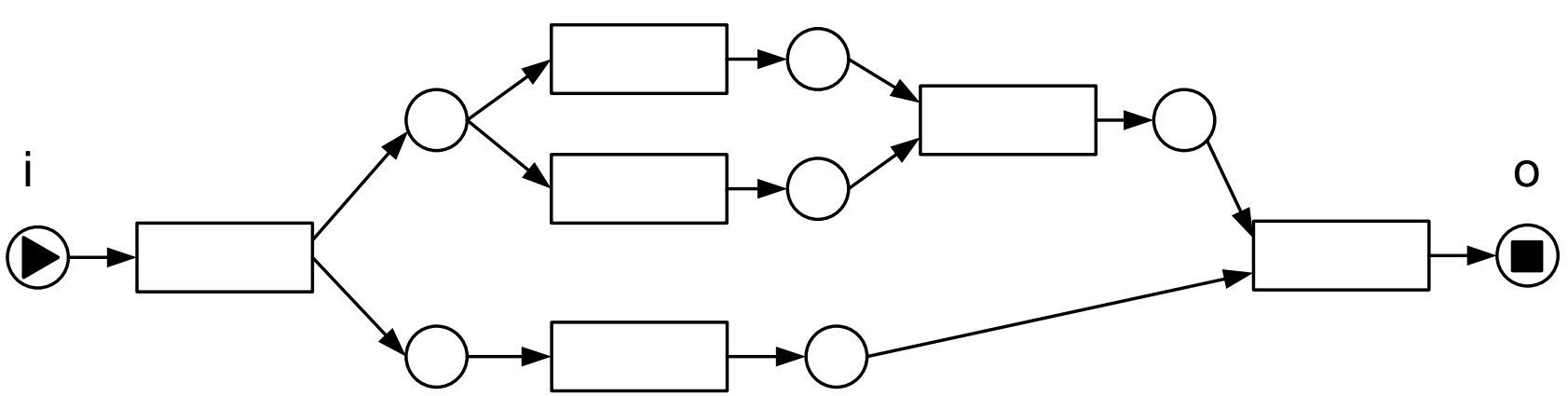
M.T. Wynn, W.M.P. van der Aalst, A.H.M. ter Hofstede and D. Edmond.
Verifying Workflows with Cancellation Regions and OR-joins: An Approach Based on Reset Nets and Reachability Analysis. BPM Center Report BPM-06-12, BPMcenter.org, 2006.

Design: YAWL Editor



Petri Net and Workflow Net

- A Petri net is a tuple **(P,T,F)** with
  - A finite set **P** of places.
  - A finite set **T** of transitions.
  - A flow relation **F** $\subseteq$ (P x T) $\cup$ (T x P)
- A Workflow net is a special Petri Net for which the following conditions hold:
  - There exists a place *i* with no incoming edge.
  - There exists a place *o* with no outgoing edge.
  - Every place and transition is located on a path from *i* o *o*

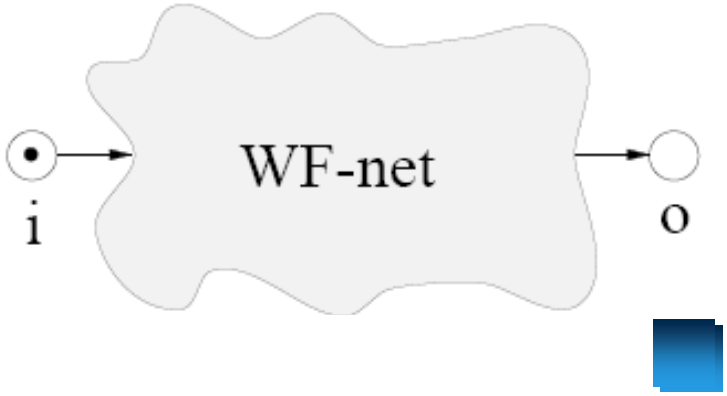


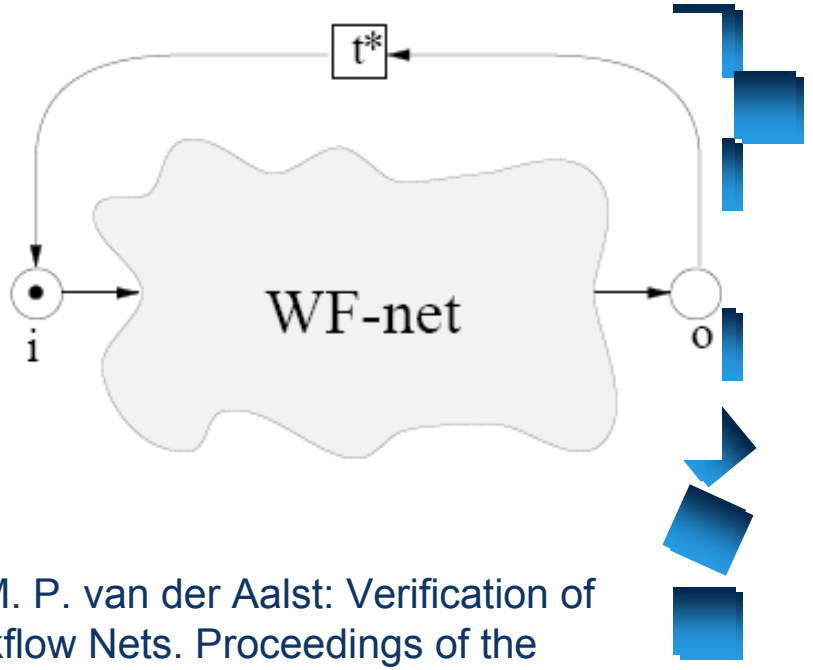
Reducing E2WF-net to Workflow Nets

- Let us replace composite and multiple instance tasks to be replace with simple tasks.
- Under these assumptions, an *E2WF-net* (**C, i, o, T, F, *split, join, rem, nofi***) can be simulated by Workflow Net **(C,T,F)**
  - The definition of marking functions is also valid for Petri Nets (and Workflow Nets)

Liveness and boundedness

- Let PN = (C,T,F) be a Workflow Net, we introduce an extended net $\underline{PN}$ = ($\underline{C}$,$\underline{T}$,$\underline{F}$) as a special Petri Net defined as:
  - $\underline{C}$ = C
  - $\underline{T}$ = T ∪ {t*}
  - $\underline{F}$ = F ∪ { (o,t*), (t*,i)}
- Let ($\underline{PN}$, i) denote a Petri Net PN with initial marking M_i
  - $M_i(x)=0, \forall x \neq i$
- ($\underline{PN}$, i) is **live** iff
  - $\forall t \in \underline{T}, \exists M', M'', M_i \rightarrow^* M' \rightarrow^t M''$
- ($\underline{PN}$, M_i) is **bounded** iff:
  - $\forall M', (M_i \rightarrow^* M') \Rightarrow (\forall c \in \underline{C}, M(c)<n)$





Wil M. P. van der Aalst: Verification of Workflow Nets. Proceedings of the Applications and Theory of Petri Nets (1997): 407-426

The Soundness Theorem / 1

Thm. For each Workflow Net PN:

PN is sound $\Leftrightarrow$ (PN, i) is live and bounded

Proof. ($\Leftarrow$)

- PN is live
  - $\exists M', M'', M_i \rightarrow^* M' \rightarrow^{t^*} M''$
  - Since *o* is the only input place of t* : $M' > M_o$
- M' is of the form (M* + o), i.e. there is at least a token in *o* plus other (possible) tokens in the remaining
- When t* fires, M'' = M* + i
- PN is bounded $\Rightarrow$ $M''(x)=0 \ \forall x \neq i \Rightarrow$ M* is identically *null*
  - Otherwise, tokens would remain aggregate in a place and their number could be ever-growing for each execution of PN.
- *Option to complete* and *Proper Completion is*, hence, proven.
- *No dead transition* follows directly from the liveness.

The Soundness Theorem / 2

Proof (Cont.). (soundness ⇒ boundedness)

The proof is given by contradiction

- PN is unbounded
  - Hence, $\exists M', M'', M_i \rightarrow^* M' \rightarrow^* M''$ where $M'' > M'$
- PN is sound
  - Hence, there exist a firing sequence σ such that $\exists M, M_i \rightarrow^* M' \rightarrow^{\sigma} M_o$
- Since $M'' > M'$, I can apply σ also to M'', thus resulting in a certain M:
  - $M' \rightarrow^{\sigma} M$ such that $M > M_o$
- In M there is a at least a token in *o* plus additional tokens in rest of the net
- Hence, PN cannot be sound: contradiction!

The Soundness Theorem / 2

Proof (Cont.). (soundness ⇒ liveness)

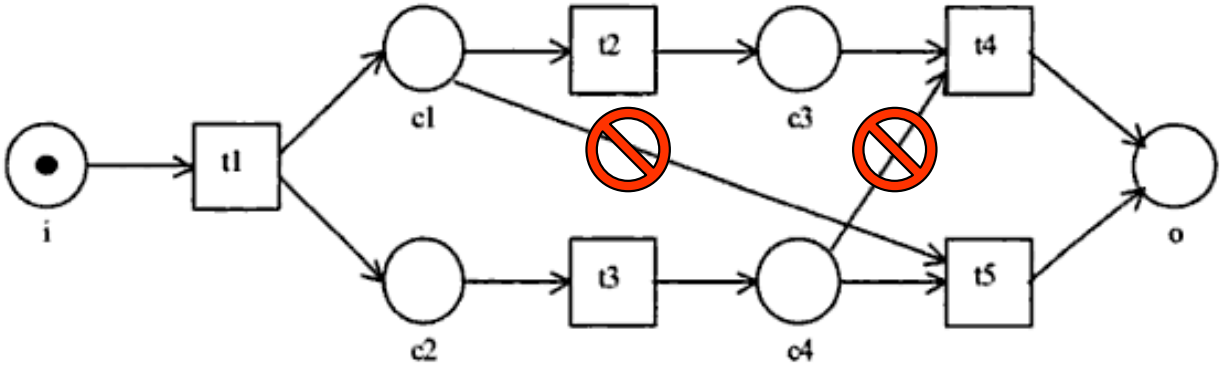
- PN is sound
  - i.e. $\forall\ t \in T, \exists\ \sigma = [t_1, t_2, \ldots, t, \ldots\ t_{n-1}, t_n]\ M_i \rightarrow^{\sigma} M_o$
  - Hence, $\forall\ t \in T, \exists M', M''\ M_i \rightarrow^{*} M' \rightarrow^{t} M''$
- Moreover, for PN the following holds by construction:
  - $M_o \rightarrow^{t^*} M_i$
- So, the liveness follows.

Free-choice nets

- For complex EWF-nets (and, hence, WF-nets), liveness and boundness are decidable is but EXPSPACE-hard
  - If the state space is not finite, the problem is not decidable.
- There exist a special WF-nets class for which the problem is more tractable:

- A WF-net (C,T,F) is **free-choice net** iff $\forall t_1,t_2 \in T$ either:
  - $\bullet t_1 = \bullet t_2$
  - $\bullet t_1 \cap \bullet t_2 = \varnothing$

Edges violating the free-choice property



Free-choice and soundness

- Non-free-choice nets are indesiderable in Business Process Modelling, since the behaviour depends on the order in which transitions enabled at the same time fire
  - Moreover, the behaviour of many non-free-choice nets is equivalent to those of corresponding free-choice.

Thm. For a free-choice WF-net it is possible to decide liveness in polynomial time.

Proof. Omitted

Thm. For a free-choice WF-net it is possible to decide soundness in polynomial time.

Proof. From the above theorem, it is possible to decide liveness and boundness in P-time. Hence, from the soundness theorem, checking soundness is P-time.

Soundness of a complete YAWL Specification

- A YAWL Specification is composed of a set of EWF-nets Q.
- In presence of OR joins, soundness is not decidable.
  - The presence of a OR join in a EWF-net at lower level can affected the behaviour of the higher-level EWF-net which has been composed in.
  - The presence of an OR join at lower level can change the behaviour of an OR join at higher level.

Thm. Let S = (Q,top,$\mathcal{T}$,map) be a YAWL workflow specification without OR-join:
$\forall$ t $\in$ $\mathcal{T}$ join(t)≠OR.
S is sound if each EWF-net N$\in$ Q is sound

W.M.P. van der Aalst and A.H.M. ter Hofstede. YAWL: Yet Another Workflow Language, Information Systems 30(4):245-275, 2005

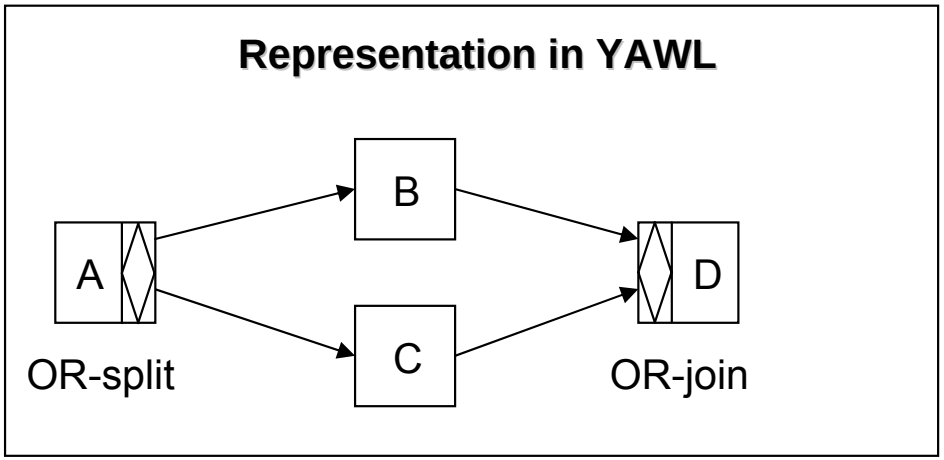
Proof. / 1

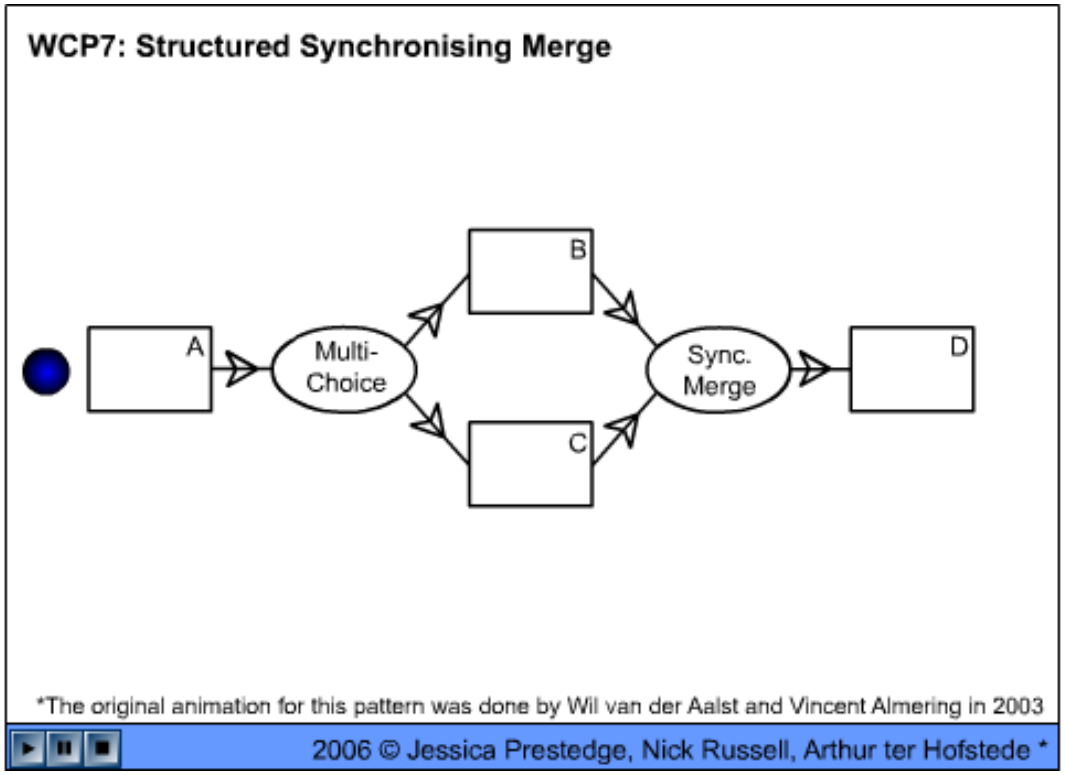
- Let us denote $H = \{(N_1,N_2) \mid N_1 \in Q \wedge N_2 \in Q \wedge \exists t \in N_1 \; map(t)=N_2\}$
- Let H^* denoting the transitive closure of H
- Let $N{\downarrow}^*$ denoting all descendents of N, including N. I.e.:
  - $N{\downarrow}^* = \{ N' \in Q \mid (N,N') \in H^* \}$
- Let S_N denoting the specification $(N{\downarrow}^*, N, \bigcup_{n \in N{\downarrow}^*} T_n$, *map)*
- Let each $N \in Q$ be sound
- The proof is by induction
  - H can be seen as a tree.
- **Base step**
  - Let us start from the leaves. For each leaf N
  - S_N is sound because $S_N = (\{N\},N,T,\varnothing)$ and N is sound

Proof. / 1

- **Induction Step**
  - Let us consider nodes N s.t.
    $\forall$ N' $\in$ N$\downarrow$*-{N} $S_{N'}$ is proven to be sound
  - S_N is sound for the following reasons:
  1. *Option to complete*
     - Since N is sound per se, the only way to block N is by blocking the completion of a task of T mapped onto some EWF-net N'
     - But, this is not possible since $S_{N'}$ is sound
  2. *No dead task*
     - Since N is sound per se, it has no dead task. Therefore, S_N cannot have dead tasks.
  3. *Proper completion*
     - For any N', when N' is completed after the activation by some task of N, since $S_{N'}$ is sound, a token is put onto $o_{N'}$ and no other token is still present in N'. Moreover, when routing is back to N, the token onto o_N is removed.
     - Therefore, when S_N completes, no token is still remaining in the net

Other constructs: Structured Synchronising Merge

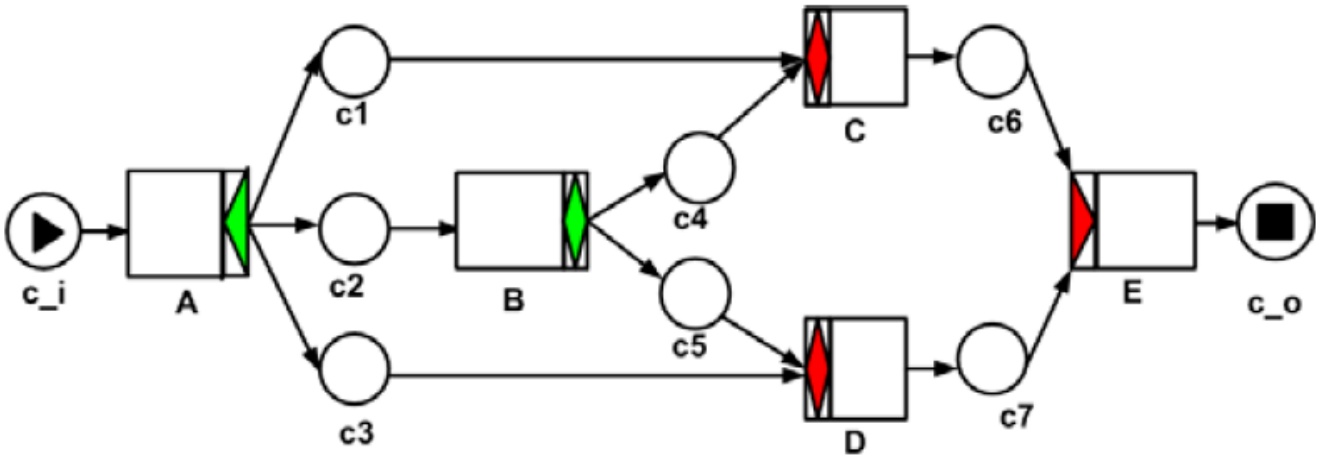




The convergence of two or more branches into a single subsequent branch such that the thread of control is passed to the subsequent branch when each active incoming branch has been enabled.

- The Structured Synchronizing Merge occurs in a structured context, i.e. there must be a single Multi-Choice construct earlier in the process model with which the Structured Synchronizing Merge is associated and it must merge all of the branches emanating from the Multi-Choice.

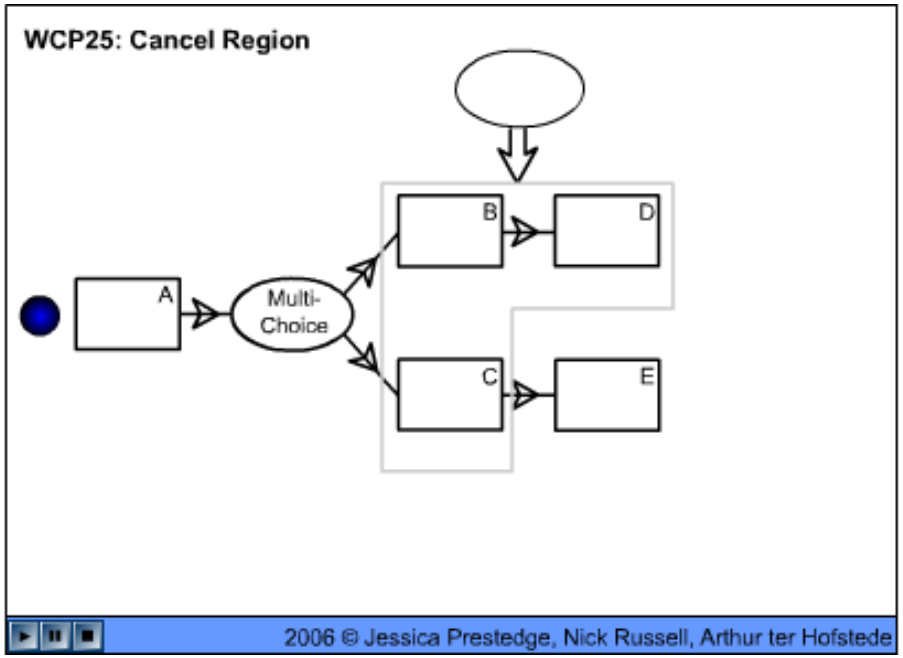
The problem of the OR-join



- Let us assume after task A, the AND-split behaviour produces tokens for conditions c2 and c3.
- The question is: Can task D fire now?
  - Task D shows a OR-join behavior.
  - To determine if/when Task D is enable, it's to find out whether a token is going to be produced for c5.
    - When reaching D, it's to analyse backwards the behaviour of the OR-split of task B.
  - If going to be produce, D will be enabled when condition c5 hosts a token
  - If no, D can fire immediately.
- Conclusion: the decision to enable an OR-join can't be made locally.

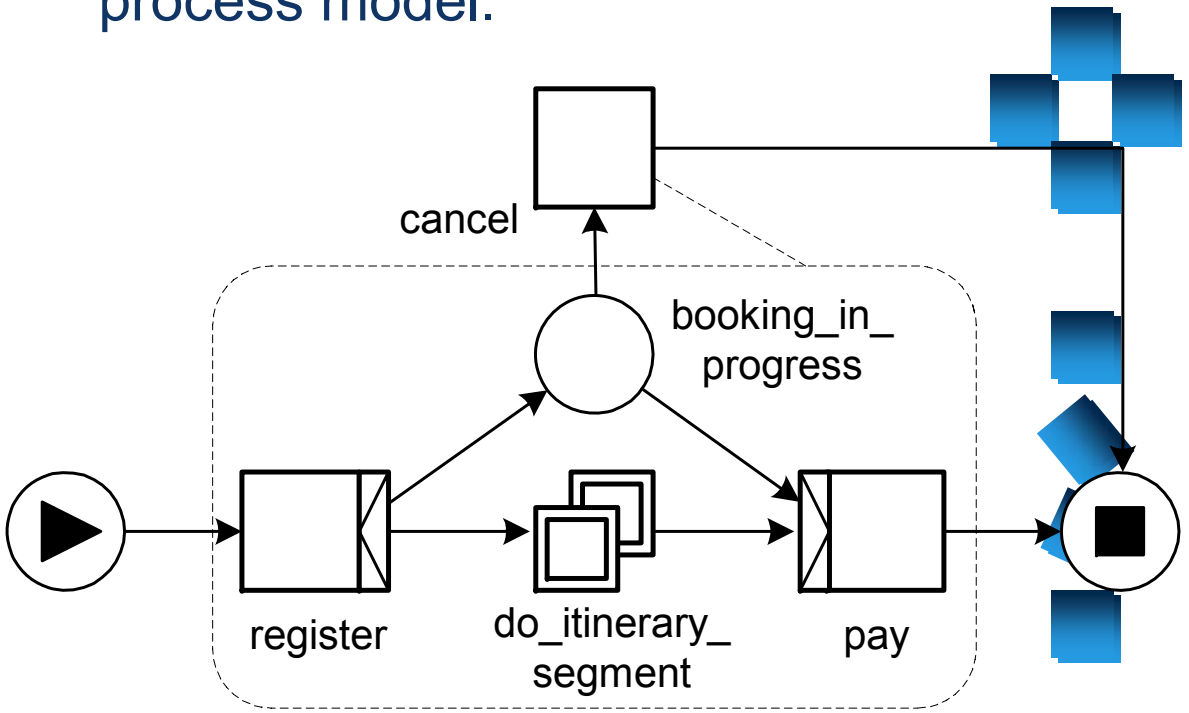
Moe Thandar Wynn, David Edmond, Wil M. P. van der Aalst, Arthur H. M. ter Hofstede: Achieving a General, Formal and Decidable Approach to the OR-Join in Workflow Using Reset Nets. Proceedings of the Applications and Theory of Petri Nets (2005), pages 423-443

Cancellation Regions



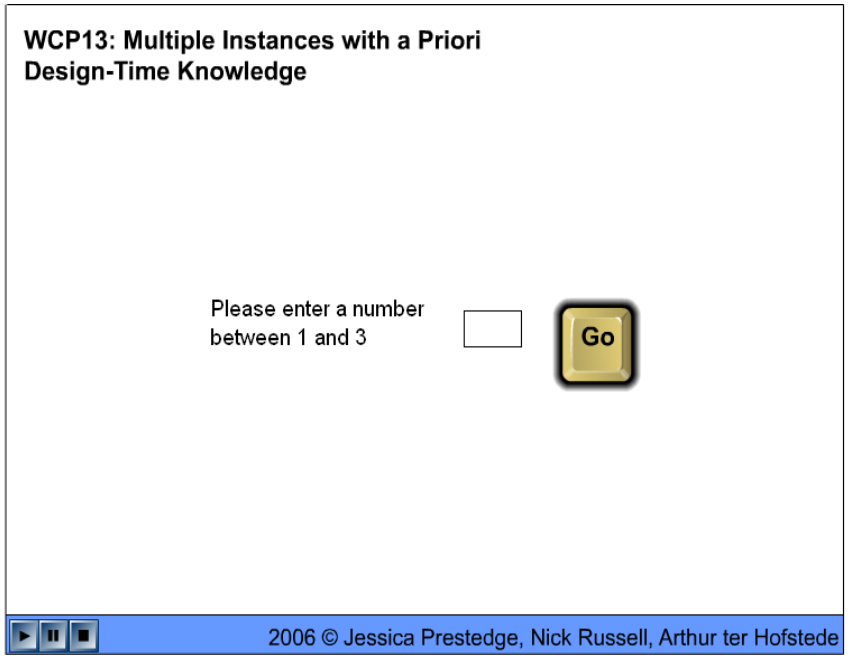
- The ability to disable a set of tasks in a process instance. If any of the tasks are already executing (or are currently enabled), then they are withdrawn. The tasks need not be a connected subset of the overall process model.

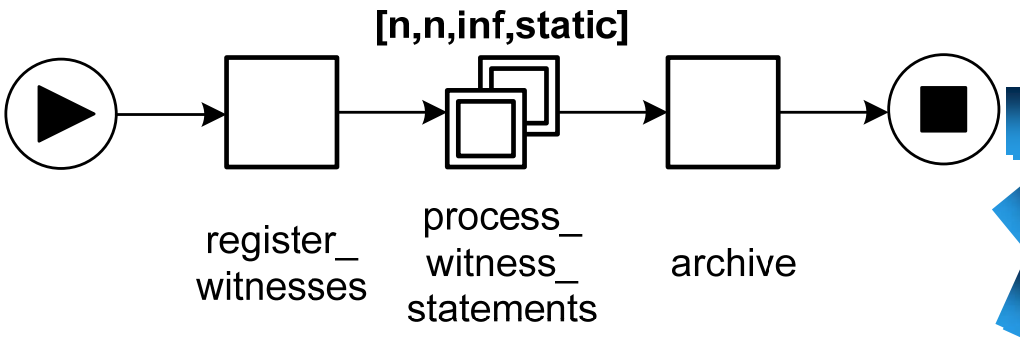
- On the right, an example is shown.
  - After the registration, task *Cancel* is enable. Its firing would cause the dotted part of the net to be removed.



Multiple Instances with a *priori* Design-Time Knowledge

- Multiple instances of a task can be created.
- The required number of instances is known at design time.
- These instances are independent of each other and run concurrently.
- It is necessary to synchronize the task instances at completion before any subsequent tasks can be triggered.

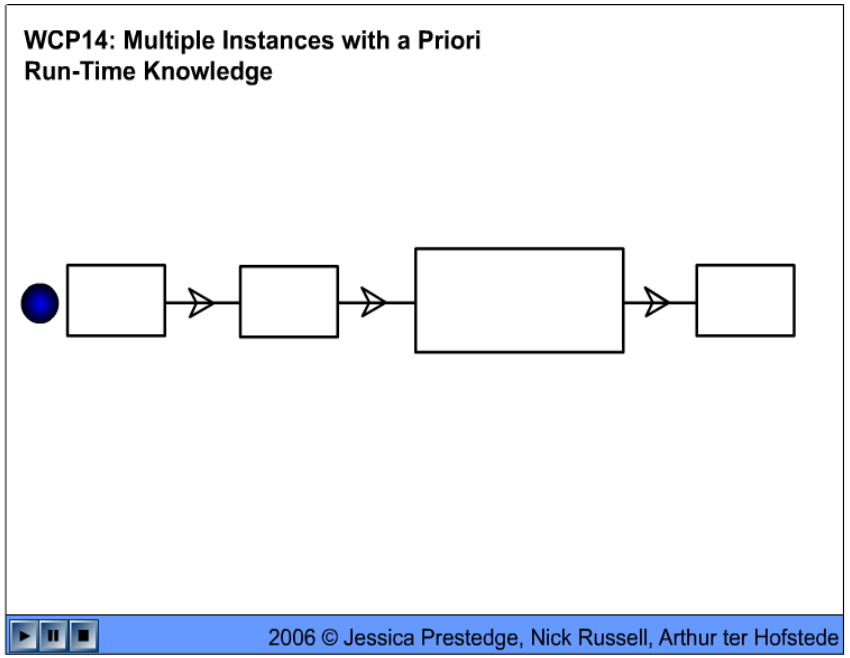


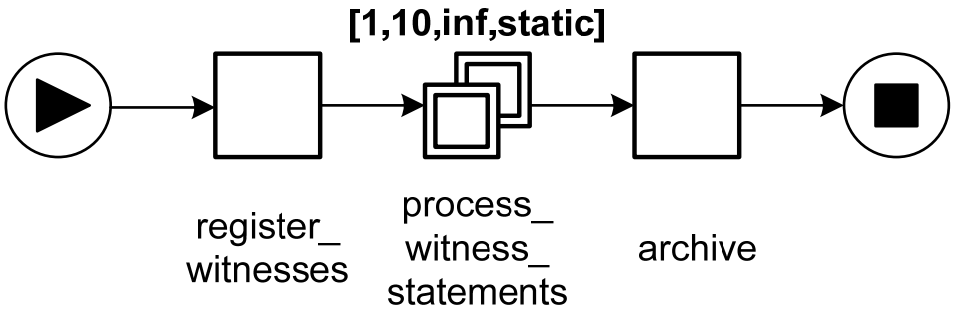


Exactly N instances of witnesses can be processed, furthermore, no more witnesses can be incorporated.

Multiple Instances with a *priori* Run-Time Knowledge

- Multiple instances of a task can be created.
- The required number of instances may depend on a number of runtime factors, including state data, resource availability and inter-process communications,
- It is known a priori the number of task instances to create.
- Once initiated, these instances are independent of each other and run concurrently.
- It is necessary to synchronize the instances at completion before any subsequent tasks can be triggered.

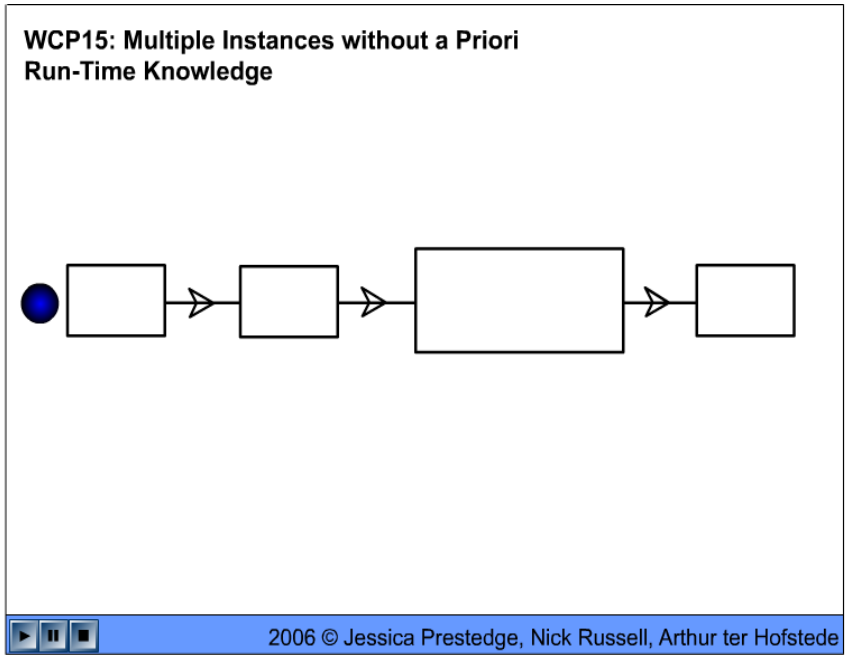


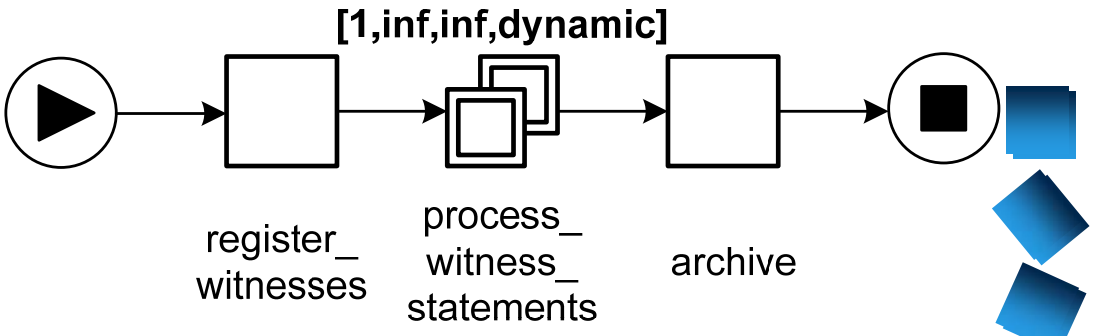


In between 1 and 10 witness statements are processed, witnesses cannot be added after registation of them has finished.

Multiple Instances without a *priori* Run-Time Knowledge

- Multiple instances of a task can be created.
- The required number of instances may depend on a number of runtime factors, including state data, resource availability and inter-process communications.
- The exact number is not known until the final instance has completed
- Whilst instances are running, it is possible for additional instances to be initiated.
- Once initiated, these instances are independent of each other and run concurrently.
- It is necessary to synchronize the instances at completion before any subsequent tasks can be triggered.





An arbitrary number of witnesses can be processed, furthermore, more witnesses can be incorporated after processing has started but before archiving.

The YAWL System

Acknowledgement

- This presentation uses slides prepared by the following people:
  - Wil van der Aalst, TUE & QUT
  - Michael Adams, QUT
  - Lachlan Aldred, QUT
  - Arthur ter Hofstede, QUT
  - Marcello La Rosa, QUT
  - Nick Russell, QUT
  - Chun Ouyang, QUT
  - Katie Shortland, AFTRS
  - Kenneth Wang, QUT
  - Petia Wohed, SU/KTH
  - Moe Wynn, QUT

Workflow 2.0

- Strong multi-perspective integrated support
- Support for the full BPM lifecycle
  - Configuration
  - Modelling
  - Pre-execution analysis (both simulation & verification)
  - Execution *(close link to modelling)*
  - Monitoring
  - Post-execution analysis
- Flexibility support
  - Exception handling
  - Declarative workflow
  - Evolving workflow
- Service-oriented architecture

YAWL Overview

- Collaboration between TU/e and QUT
- Based on Workflow Patterns Initiative
- YAWL: 2002 – newYAWL: 2007
- System development
  - Open source (currently LGPL)
  - Governed by YAWL Foundation
- Main publications
  - W.M.P. van der Aalst and A.H.M. ter Hofstede. YAWL: Yet Another Workflow Language, Information Systems 30(4):245-275, 2005
- URLs:
  - www.yawlfoundation.org (research)
    - Manual of the YAWL system available on the web site.
  - www.sourceforge.net/projects/yawl (system)

YAWL Highlights

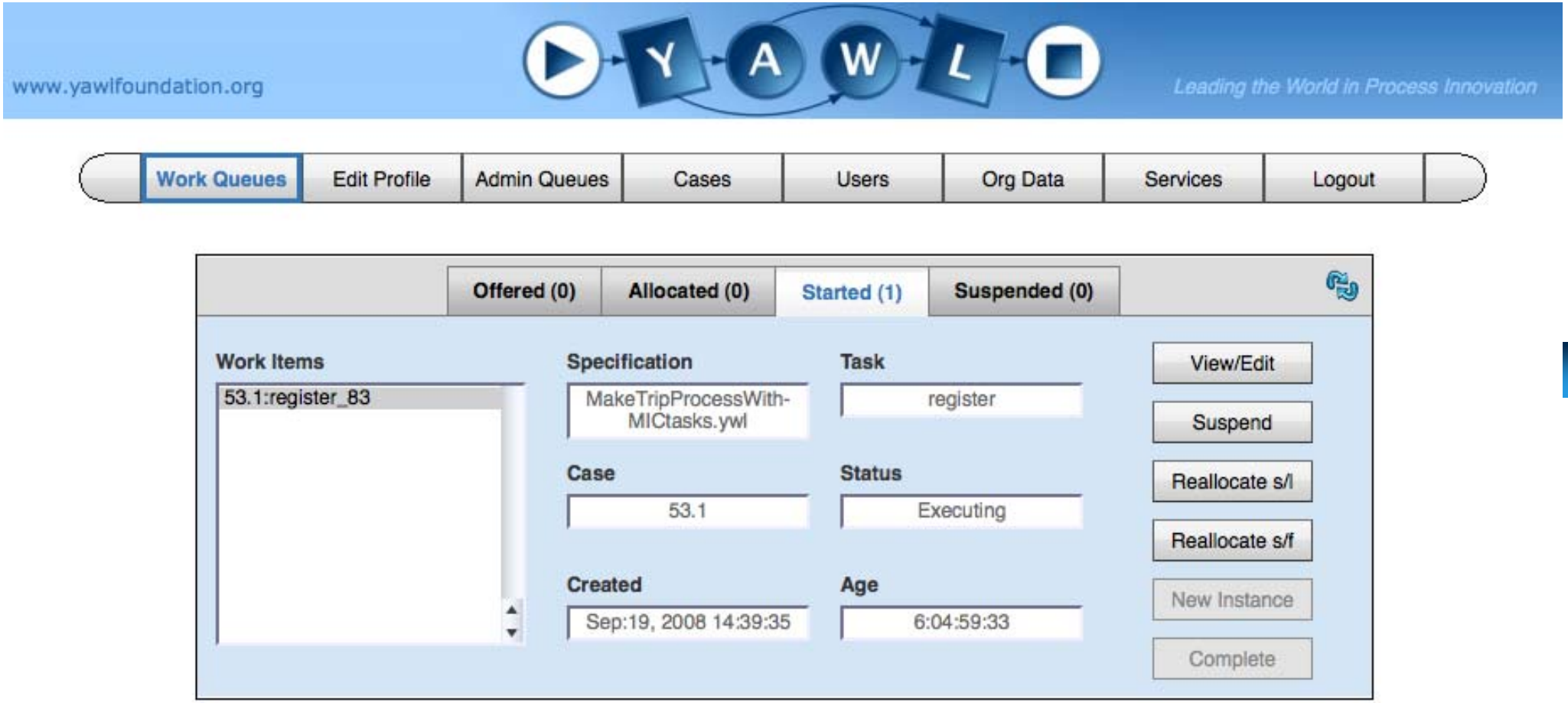
- **YAWL**
  - Based on the (old) control-flow patterns
  - Extends Petri nets
  - Formal Foundation
  - Verification
  - Dynamic workflow
  - Declarative workflow
  - Exception handling

- **YAWL System**
  - Open source
  - Service oriented architecture
  - Production class
  - Comprehensive support for control-flow and resource patterns
  - Strong support for flexibility
  - Link to ProM for post-execution analysis and simulation

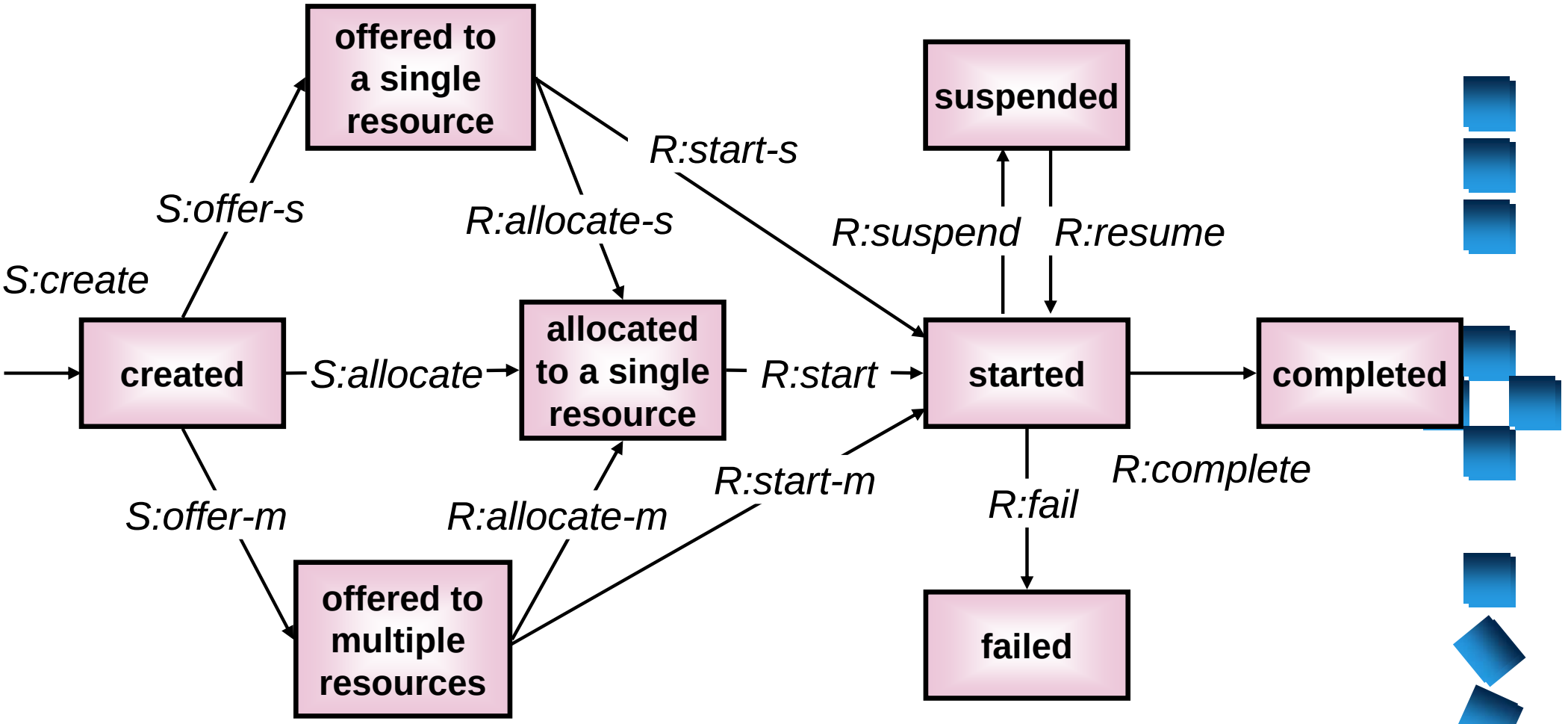
The YAWL System

- Engine, Graphical Editor
- Custom YAWL Services
  - (e.g. Declare, Worklets, Digital Signature)
- First release November 2003, Open Sourced in May 2004
- YAWL 2.0 has been released in early 2009.
- Perspectives:
  - Control-flow: comprehensive support
  - Data: using XML technologies
  - Resource: comprehensive support
  - Operational: tasks can be linked to web services
- Service-oriented architecture
- Link to ProM for process mining and analysis.

Work Lists: Default View

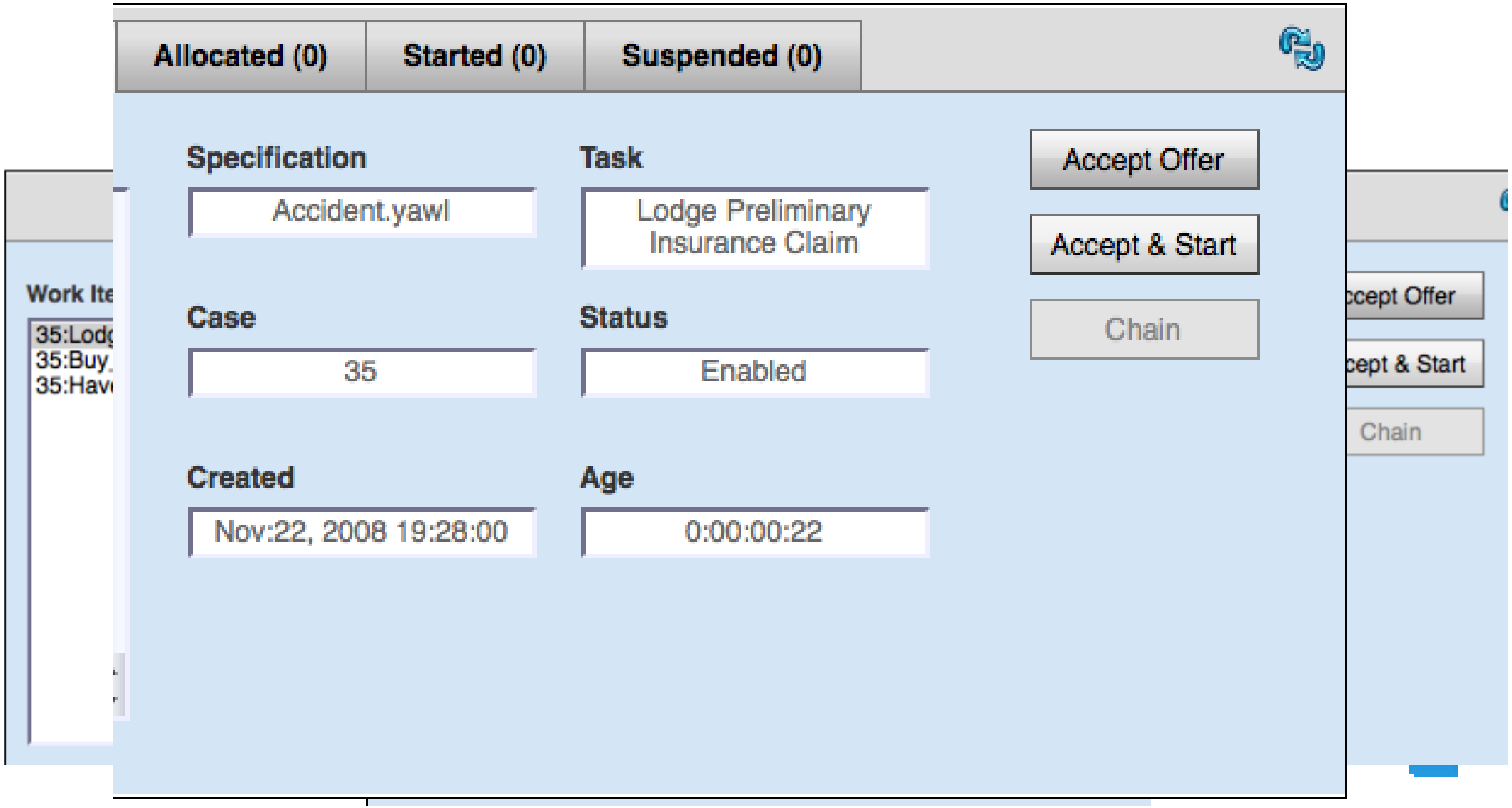


Work Item Lifecycle

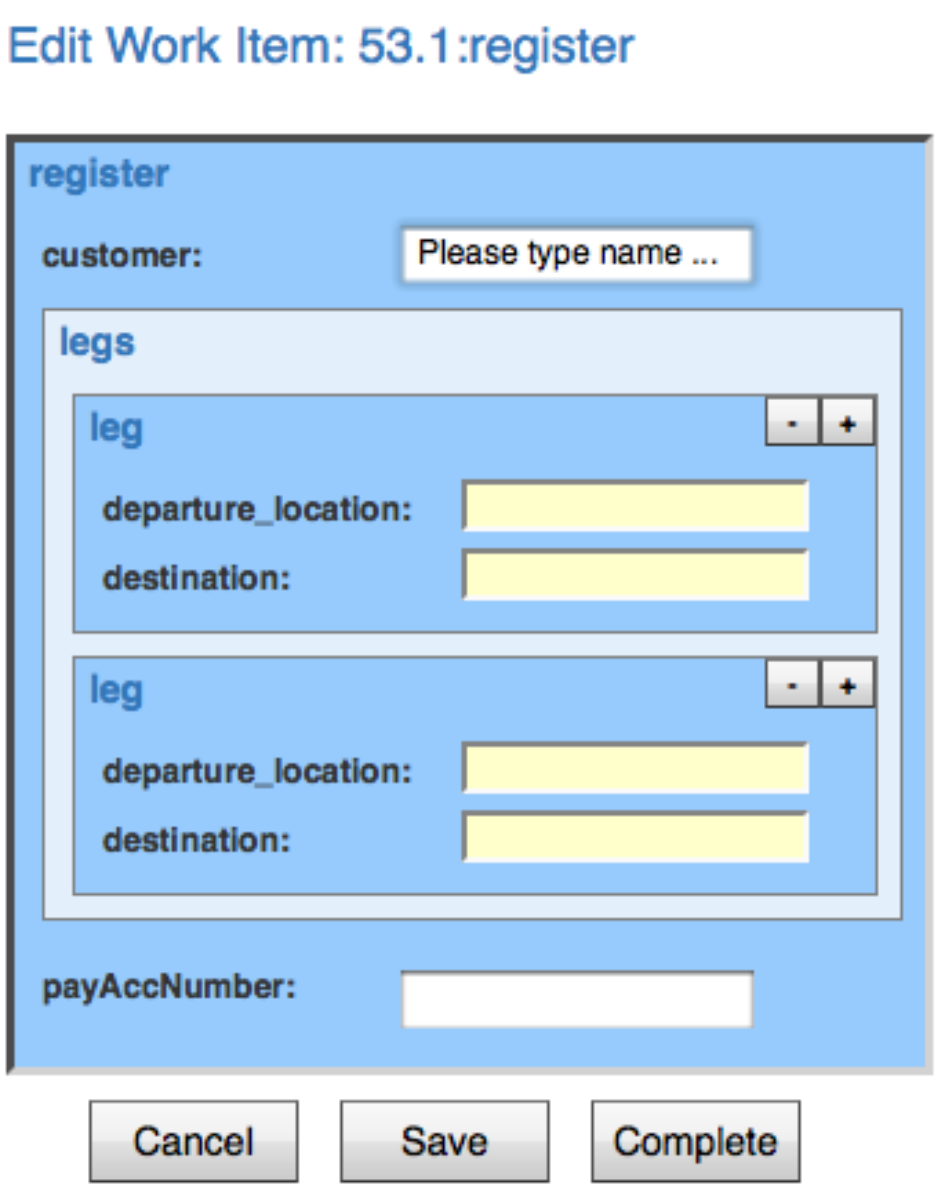


Nick Russell, Wil M. P. van der Aalst, Arthur H. M. ter Hofstede, David Edmond: Workflow Resource Patterns: Identification, Representation and Tool Support. CAiSE 2005: 216-232

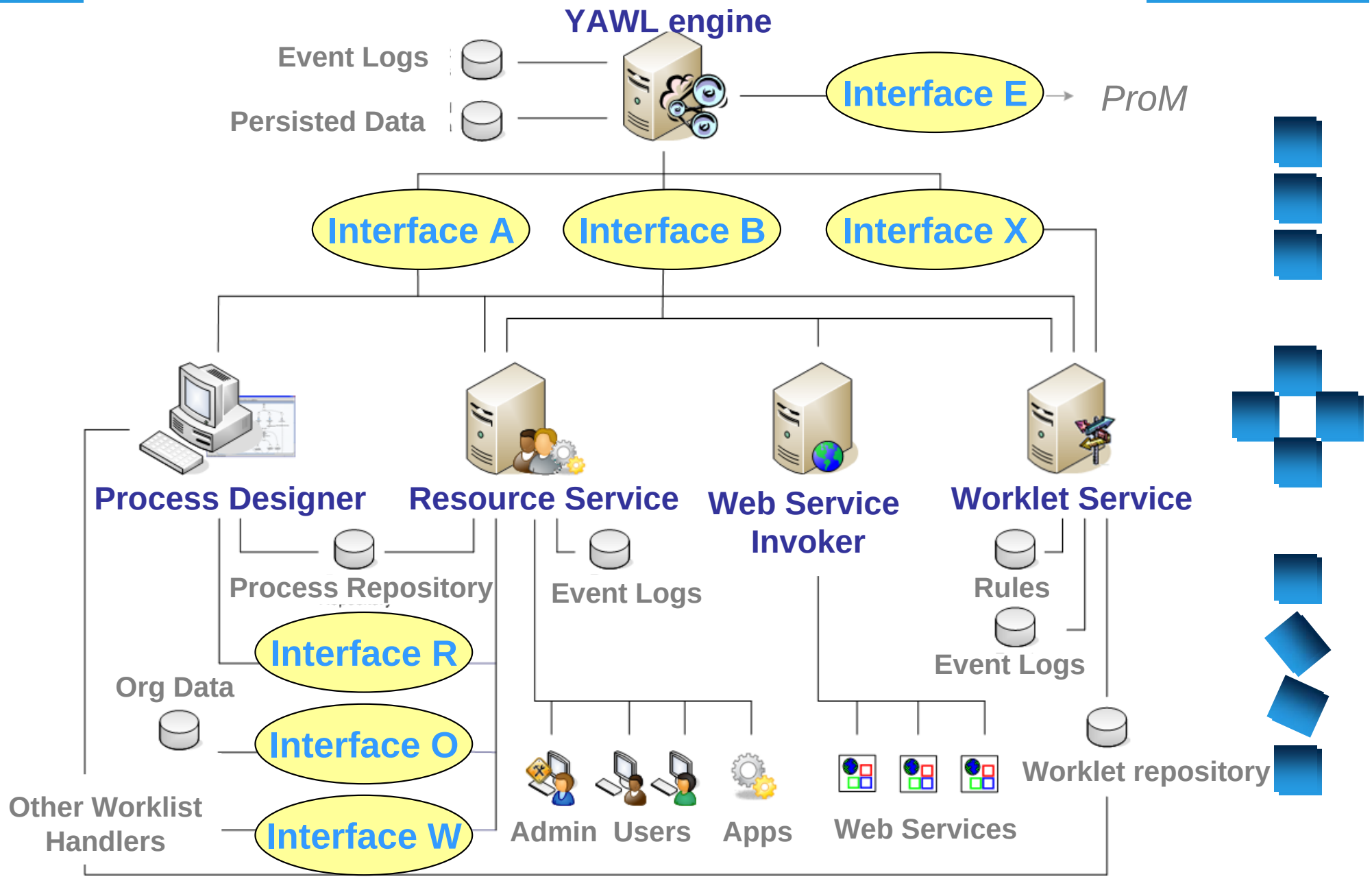
YAWL Engine – Worklist Screenshot



Native use of XML: Generation of XForms



Service Oriented Architecture

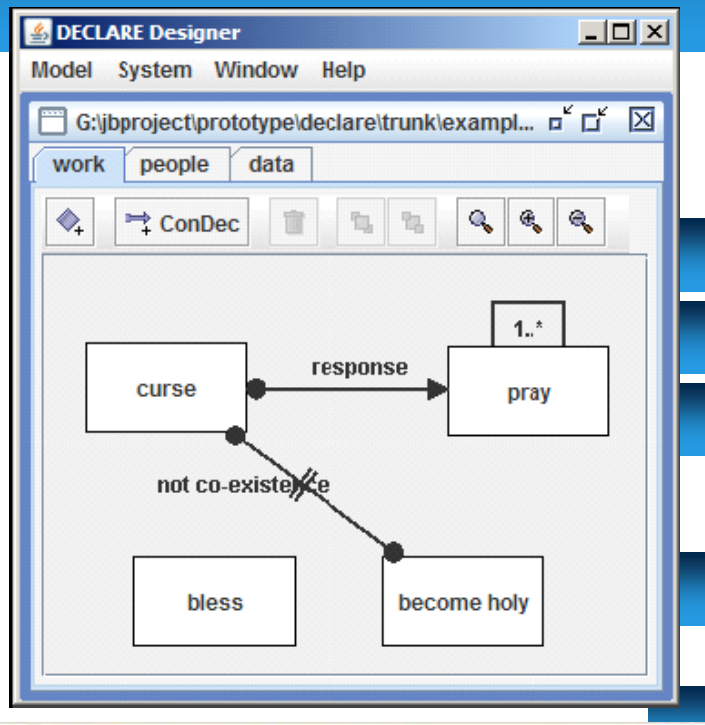


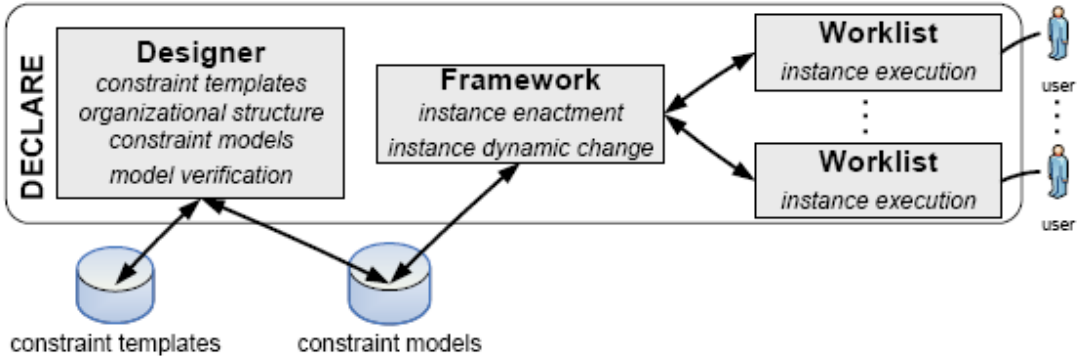
Advanced topics: Exception Handling

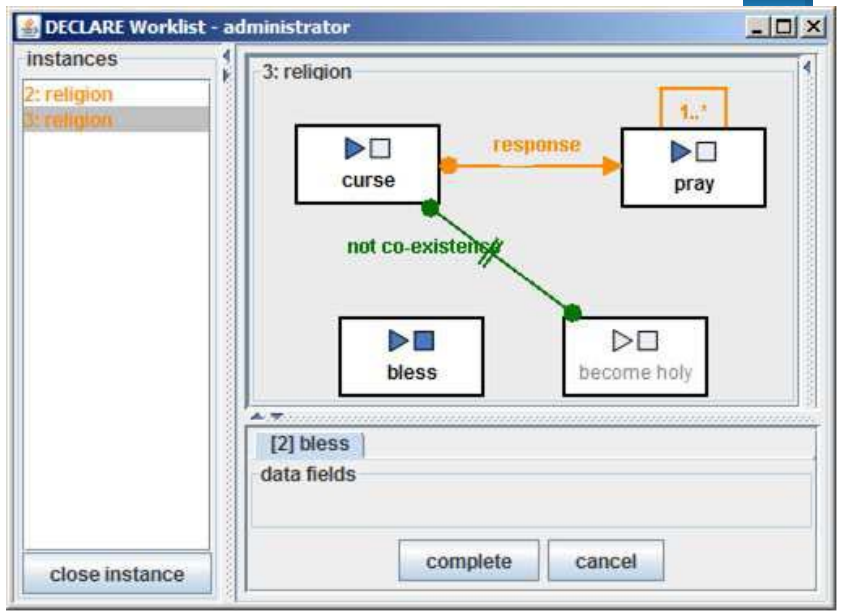
- Processes are defined thus being executed under certain circumstances.
  - When cases are instantiated, the process specifications are made s.t. all work items created are executable according to the expected state of the working environment in that point in time.
- What happens if some unforeseen contingencies occur? The state changes unexpectedly!
  - They may prevent running cases from being completed successfully.
- Some rules need to be defined to *restructure* the case schema.
  - Some BPMS are based on rules ECA: Event-Condition-Action
    - Upon an **event** E, under a certain **condition** C, some **actions** A are executed (to handle with the exceptional event).
  - Alternatively, a responsible domain-expert person may be in charge of manually modifying the schema
- The YAWL System is provided with some techniques for exception handling, which relies on the concept of **Worklet**
  - An extensible repertoire of self-contained sub-processes and associated selection rules.
  - Worklets are associated to some tasks (and, hence, to work items).
  - At run-time, according to the selection rules, work items are executed via one of worklets.

Advanced topics: Declare

- DECLARE is a fully functional constraint-based WFMS.
  - Semantics are grounded in Linear Temporal Logic (LTL).
- It allows for creating, verifying, executing and dynamically changing constraint-based process models.

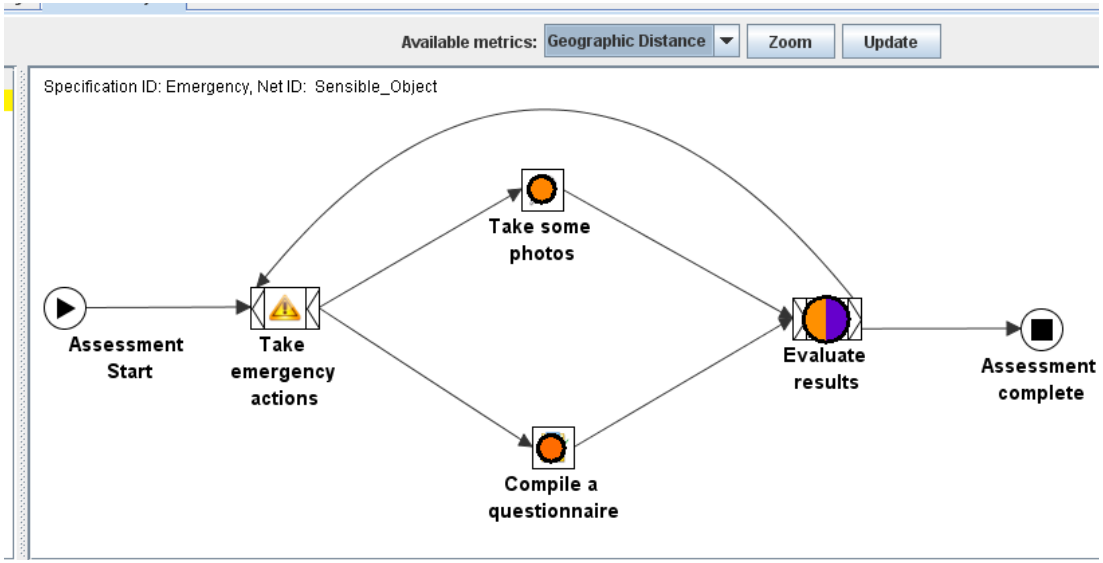


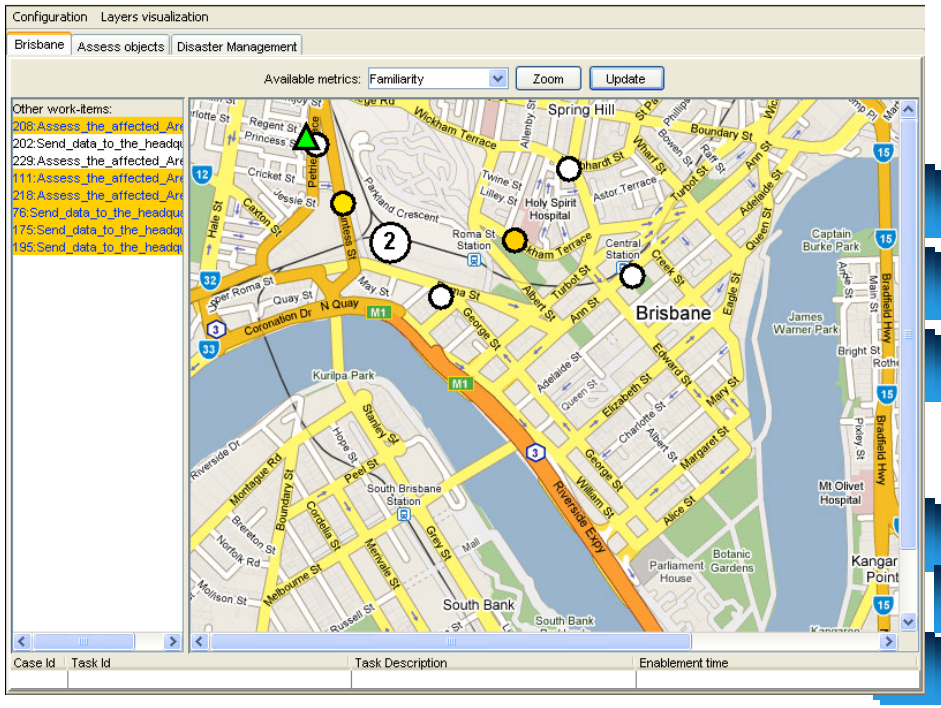




Advanced topics: Alternative View

- Typically, work lists present items in simple textual lists
  - minimal context
- At times, a resource may have a large number of work items on offer
- Providing meaningful context greatly assists the decision of which work item to perform as next
- Our approach integrates visual context into YAWL's work-list handler





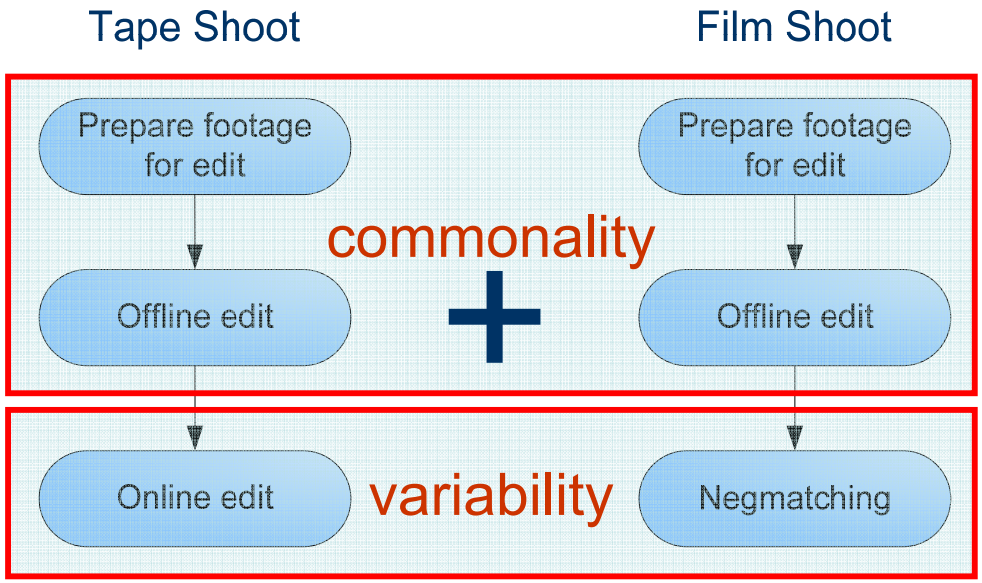
- **Map metaphor:** Work items are placed as dots on maps in meaningful positions
- **Metric metaphor:** Dots are coloured according to some execution metrics.
- Participants can switch at any time to the most significant metric for the specific setting.

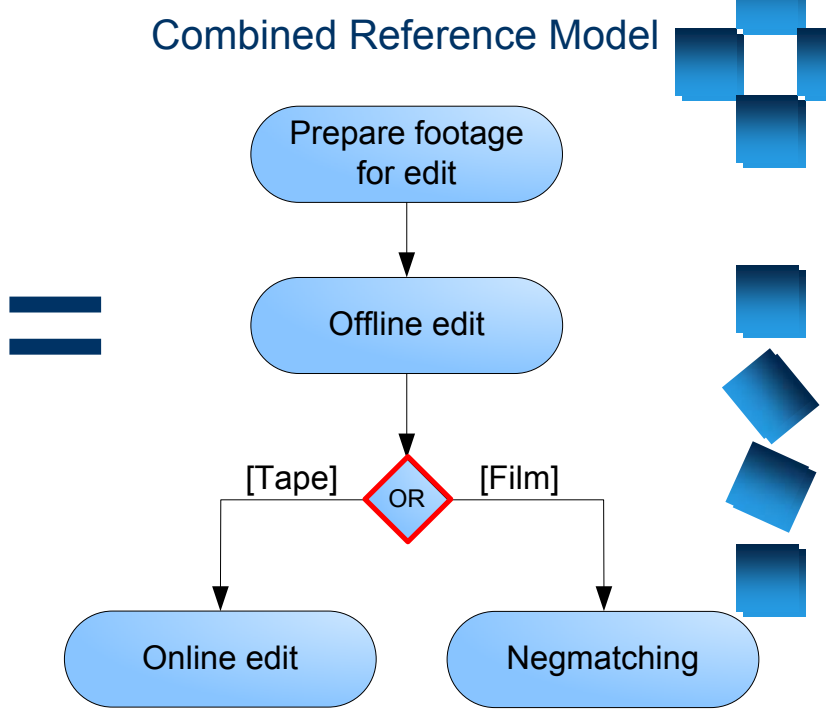
M. de Leoni, W. M. P. van der Aalst, A.H.M. ter Hofstede, "Visual Support for Work Assignment in Process-aware Information Systems", In Proc. of the 6th International Conference on Business Process Management (BPM 2008), Milan, Italy, 1-4 September 2008.

Advanced topics: Process configuration

Repository of business process models capturing practices in a domain, which can be configured in a specific setting leading to individualized process models:

- increased reuse of proven practices,
- reduced modeling effort.





Advanced topics: Process Mining

- Process Mining concerns the analysis of business processes on the basis of event logs.
- It extracts knowledge from the events of a information system.
- It aims to increase the process modelling by techniques and tools to disclosure the structure of processm, organisation models, data.
- Three main classes exist for process mining, based on the presence or assence of an a-priori model:
  - *Discovering*: No a priori model. It needs to be designed through logging data. Recently, techniques have been explorer to consider data, resources, time aspects.
  - *Conformance*: There is an a-priori model and it has to be confronted with an event log to look for possible discrempancies with the pre-existing model.
  - *Extension*: A-priori models are existing. The goal is to enrich them, thus considering new viewpoints (e.g., to enclose performance data).
- The YAWL System is able to export logging data to ProM for mining
  - ProM offers a wide variety of process mining techniques
  - It allows for the import/export of many formats.
  - It provides advanced visualization and verification capabilities.

W.M.F van der Aalst, et al. “ProM 4.0: Comprehensive Support for Real Process Analysis.” ICATPN 2007: 28th International Conference on Applications and Theory of Petri Nets and Other Models of Concurrency.

yawlbook.com