TU/e technische universiteit eindhoven

Towards a Workflow Language based on XML, Petri Nets, and Workflow Patterns

Wil van der Aalst
Eindhoven University of Technology, Faculty of Technology Management
Department of Information and Technology, P.O.Box 513, NL-5600 MB,
Eindhoven, The Netherlands.

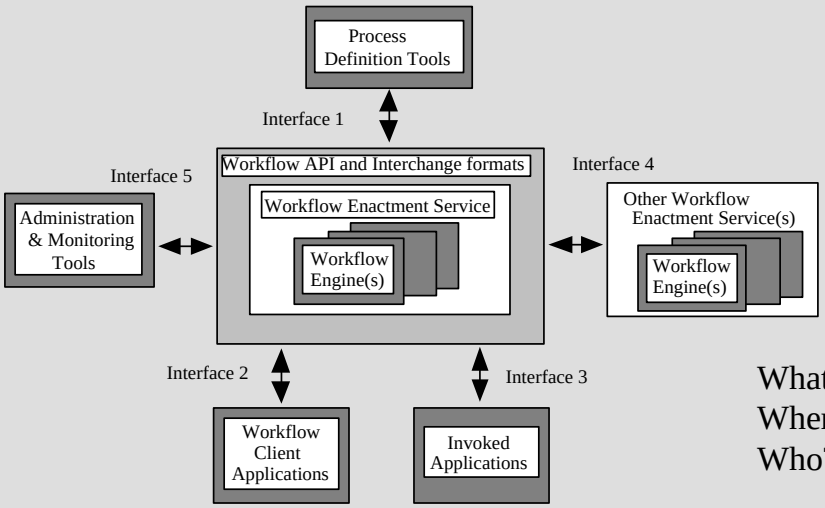
TU/e technische universiteit eindhoven

Outline

- Introduction
  - Workflow management
  - Limitations of existing systems
- Part I: Workflow patterns
  - Examples
  - Evaluation of systems
- Part II: XRL
  - XML based Language
  - Petri net Semantics
  - Architecture
- Conclusion

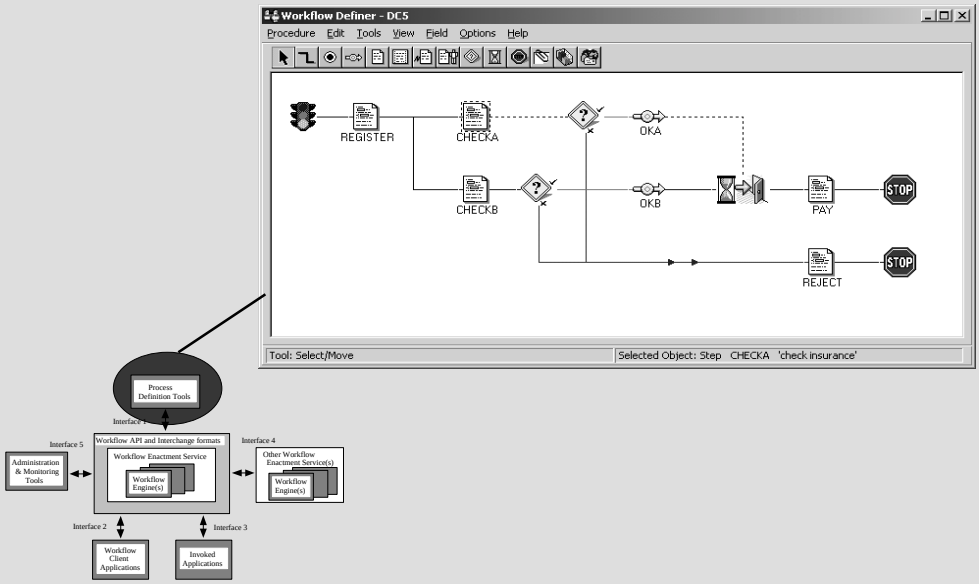
TU/e technische universiteit eindhoven

A one minute introduction to workflow management (systems) …

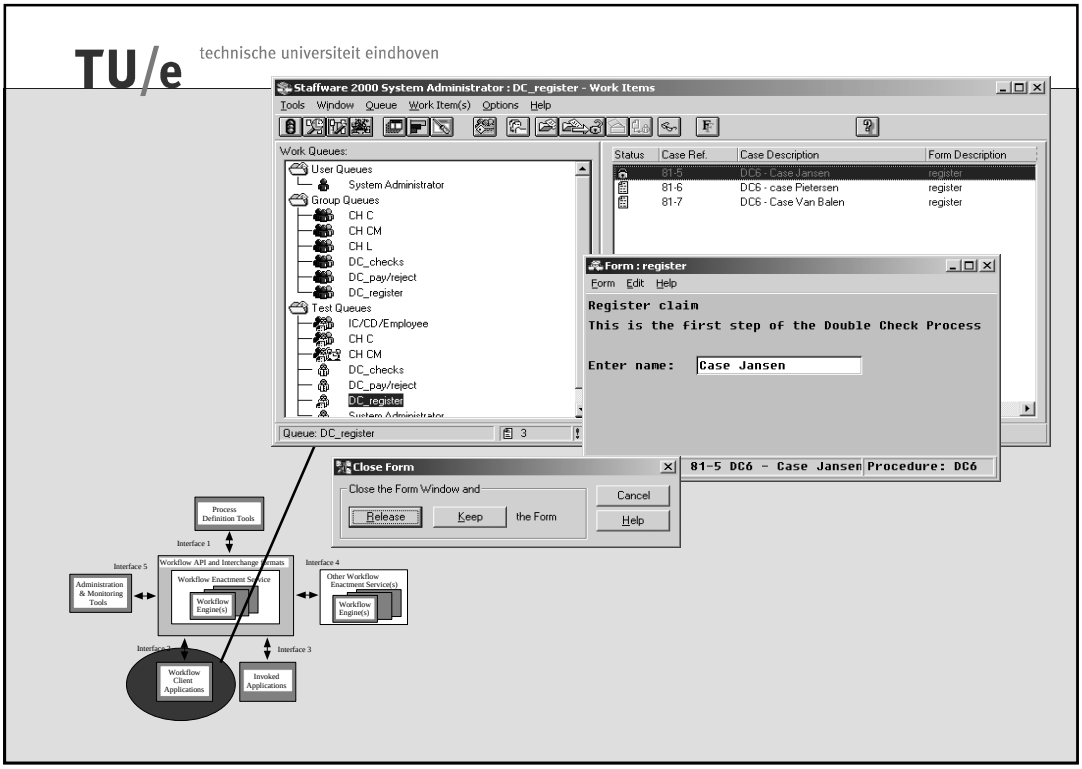


What?
When?
Who?

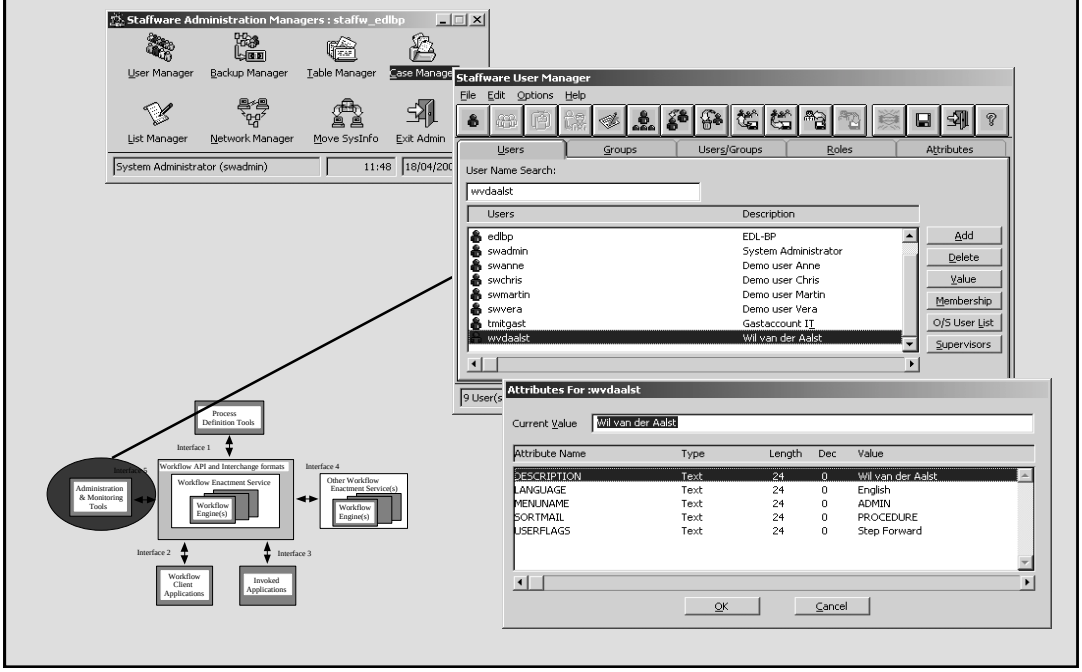
TU/e technische universiteit eindhoven



TU/e technische universiteit eindhoven



TU/e technische universiteit eindhoven



TU/e technische universiteit eindhoven

Contemporary WFM systems

- Features:
  - generic support for operational processes
  - adaptable processes, no programming, graphical, etc.
- Limitations:
  - difficulties supporting complex processes (lack of expressive power) — *Part I: Patterns*
  - limited run-time flexibility
  - limited support for interorganizational workflows
  - limited support for analysis

Part II: XRL

TU/e technische universiteit eindhoven

Part I: workflow patterns

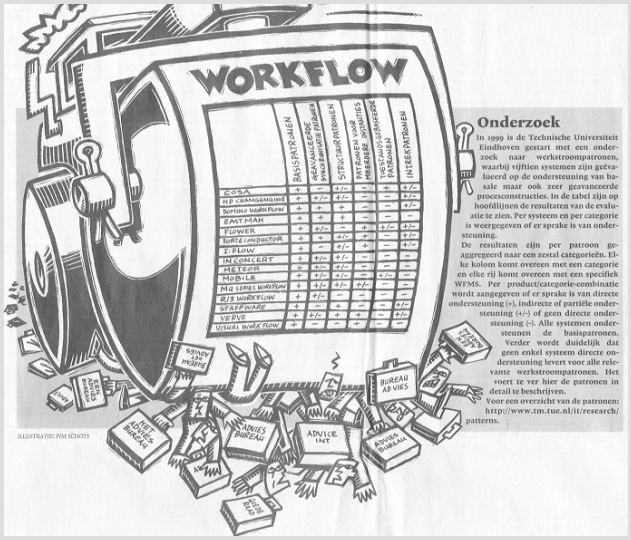
Joint work with Arthur ter Hofstede (QUT), Bartek Kiepuszewski (QUT), Alistair Barros (UQ), Oscar Ommert (EUT), Ton Pijpers (ATOS), et al.

http://www.tm.tue.nl/it/research/patterns/

TU/e technische universiteit eindhoven

Workflow patterns

- The academic response
- A quest for the basic requirements
- 20 basic patterns
- 16 systems
- Joint work with QUT, ATOS, etc.



TU/e technische universiteit eindhoven

Categories of patterns

Basic Control Flow Patterns
- Pattern 1 (Sequence)
- Pattern 2 (Parallel Split)
- Pattern 3 (Synchronization)
- Pattern 4 (Exclusive Choice)
- Pattern 5 (Simple Merge)

Advanced Branching and Synchronization Patterns
- Pattern 6 (Multi-choice)
- Pattern 7 (Synchronizing Merge)
- Pattern 8 (Multi-merge)
- Pattern 9 (Discriminator)

Structural Patterns
- Pattern 10 (Arbitrary Cycles)
- Pattern 11 (Implicit Termination)

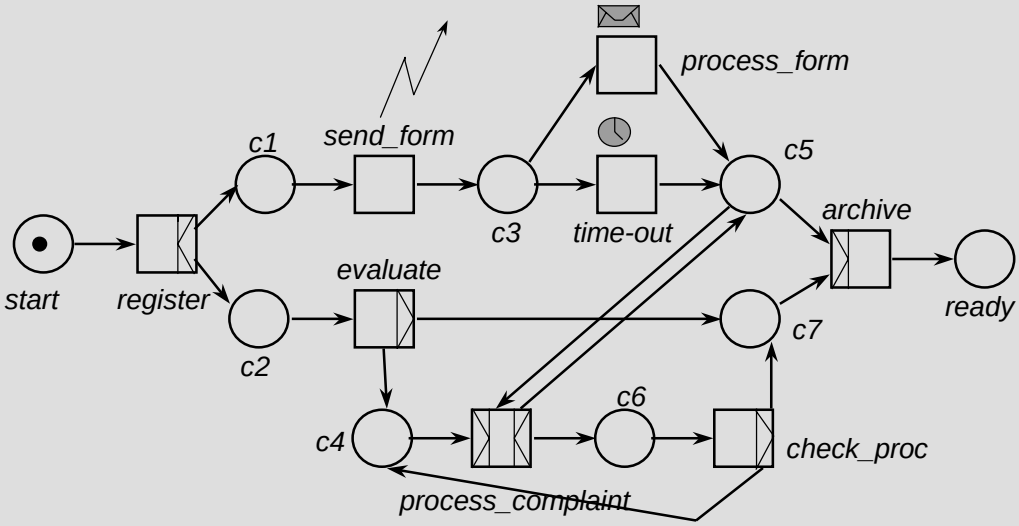
Patterns involving Multiple Instances
- Pattern 12 (Multiple Instances Without Synchronization)
- Pattern 13 (Multiple Instances With a Priori Design Time Knowledge)
- Pattern 14 (Multiple Instances With a Priori Runtime Knowledge)
- Pattern 15 (Multiple Instances Without a Priori Runtime Knowledge)

State-based Patterns
- **Pattern 16 (Deferred Choice)**
- Pattern 17 (Interleaved Parallel Routing)
- **Pattern 18 (Milestone)**

Cancellation Patterns
- Pattern 19 (Cancel Activity)
- Pattern 20 (Cancel Case)

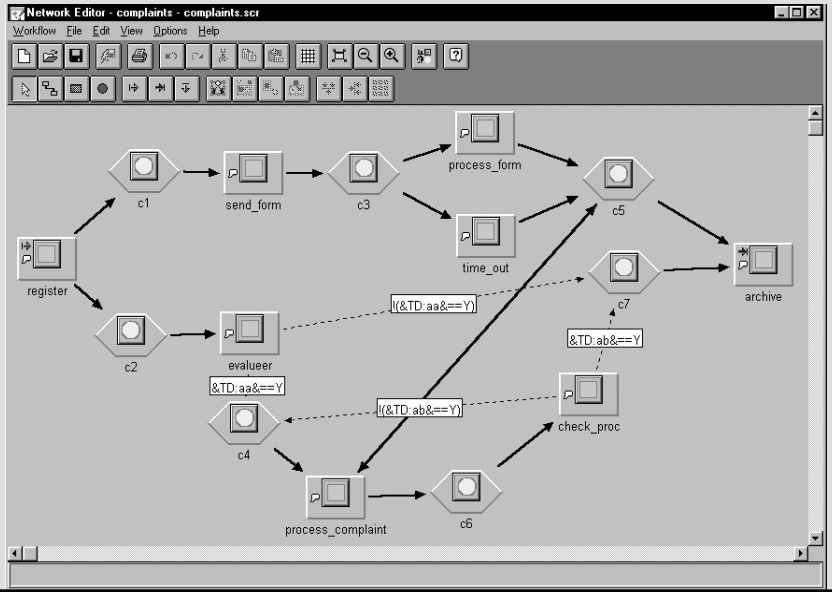
TU/e technische universiteit eindhoven

Example process: Complaints handling



TU/e technische universiteit eindhoven

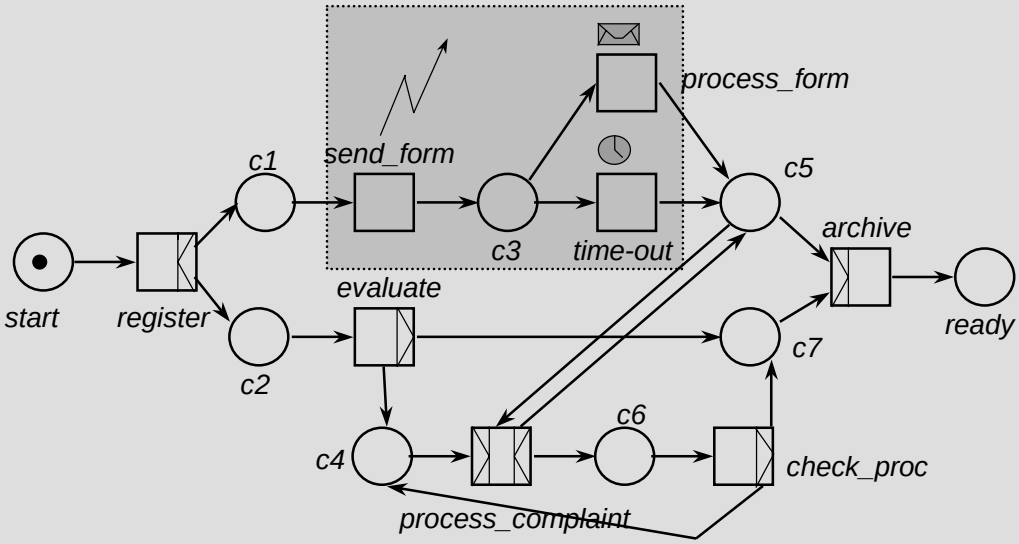
The process can be handled by COSA ….



But not by many others …

TU/e technische universiteit eindhoven

Workflow pattern 16: Deferred Choice



TU/e technische universiteit eindhoven

Pattern 16 (Deferred Choice)

Description A point in the workflow process where one of several branches is chosen. In contrast to the XOR-split, the choice is not made explicitly (e.g. based on data or a decision) but several alternatives are offered to the environment. However, in contrast to the AND-split, only one of the alternatives is executed. This means that once the environment activates one of the branches the other alternative branches are withdrawn. It is important to note that the choice is delayed until the processing in one of the alternative branches is actually started, i.e. the moment of choice is as late as possible.

Synonyms External choice, implicit choice, deferred XOR-split.

Examples

- At certain points during the processing of insurance claims, quality assurance audits are undertaken at random by a unit external to those processing the claim. The occurrence of an audit depends on the availability of resources to undertake the audit, and not on any knowledge related to the insurance claim. Deferred Choices can be used at points where an audit might be undertaken. The choice is then between the audit and the next activity in the processing chain. The audit activity triggers the next activity to preserve the processing chain.
- Consider activity A in Figure 11 to represent the activity *send_questionnaire*, and activities B and C , the activities *time_out* and *process_questionnaire* respectively. The activity *time_out* requires a time trigger, while the activity *process_questionnaire* is only to be executed if the complainant returns the form that was sent (hence an external trigger is required for its execution). Clearly, the moment of choice between *process_questionnaire* and *time_out* should be as late as possible. If this choice was modeled as an explicit XOR-split (Pattern 4), it is possible that forms which are returned in time are rejected, or cases are blocked if some of the forms are not returned at all.
- After receiving products there are two ways to transport them to the department. The selection is based on the availability of the corresponding resources. Therefore, the choice is deferred until a resource is available.
- Business trips require approval before being booked. There are two ways to approve a task. Either the department head approves the trip (activity A1) or both the project manager (activity A21) and the financial manager (activity A22) approve the trip. The latter two activities are executed sequentially and the choice between A1 on the one hand and A21 and A22 on the other hand is implicit, i.e., at the same time both activity and activity are offered to the department head and project manager respectively. The moment one of these activities is selected, the other one disappears.

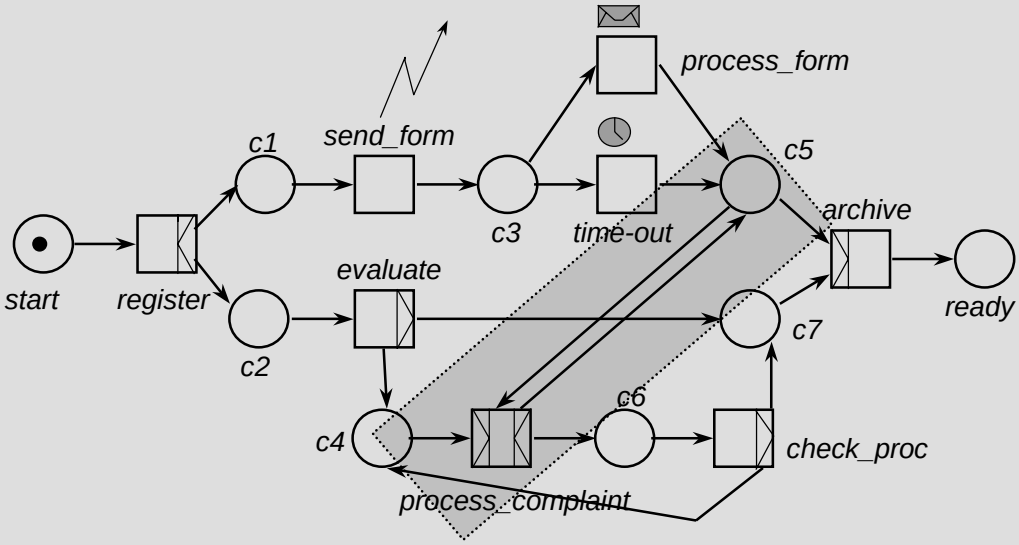
Problem Many workflow management systems support the XOR-split described in Pattern 4 but do not support the deferred choice. Since both types of choices are desirable (see examples), the absence of the deferred choice is a real problem.

Implementation

- COSA is one of the few systems that directly supports the deferred choice. Since COSA is based on Petri nets it is possible to model implicit choices as indicated in Figure 11(A). Some systems offer partial support for this pattern by offering special constructs for a deferred choice between a user action and a time out (e.g., Staffware) or two user actions (e.g., FLOWer).
-Assume that the workflow language being used supports cancellation of activities through either a special transition (for example Staffware, see Pattern 19 (Cancel Activity)) or through an API (most other engines). Cancellation of an activity means that the activity is being removed from the designated worklist as long as it has not been started yet. The deferred choice can be realized by enabling all alternatives via an AND-split. Once the processing of one of the alternatives is started, all other alternatives are canceled. Consider the deferred choice between and in Figure 11 (Workflow A). After , both and are enabled. Once is selected/executed, activity is canceled. Once is selected/executed, activity is canceled. Workflow A of Figure 12 shows the corresponding workflow model. Note that the solution does not always work because and can be selected/executed concurrently.
- Another solution to the problem is to replace the deferred choice by an explicit XOR-split, i.e. an additional activity is added. All triggers activating the alternative branches are redirected to the added activity. Assuming that the activity can distinguish between triggers, it can activate the proper branch. Consider the example shown in Figure 11. By introducing a new activity after and redirecting triggers from and to , the implicit XOR-split can be replaced by an explicit XOR-split based on the origin of the first trigger. Workflow B of Figure 12 shows the corresponding workflow model. Note that the solution moves part of the routing to the application or task level. Moreover, this solutions assumes that the choice is made based on the type of trigger.

TU/e technische universiteit eindhoven

Workflow pattern 18: Milestone



TU/e technische universiteit eindhoven

Pattern 18 (Milestone)

Description The enabling of an activity depends on the case being in a specified state, i.e. the activity is only enabled if a certain milestone has been reached which did not expire yet. Consider three activities named A, B, and C. Activity is A only enabled if activity B has been executed and C has not been executed yet, i.e. A is not enabled before the execution of B and is not enabled after the execution of C. Figure 16 illustrates the pattern. The state in between B and C is modeled by place M. This place is a milestone for A. Note that A does not remove the token from M: It only tests the presence of a token.

Synonyms Test arc, deadline (cf. [JB96]), state condition, withdraw message.

Examples

- In a travel agency, flights, rental cars, and hotels may be booked as long as the invoice is not printed.
- A customer can withdraw purchase orders until two days before the planned delivery.
- A customer can claim air miles until six months after the flight.
- The construct involving activity *process_complaint* and place *c5* shown in Figure 15.

Problem The problem is similar to the problem mentioned in Pattern 16 (Deferred Choice): There is a race between a number of activities and the execution of some activities may disable others. In most workflow systems (notable exceptions are those based on Petri nets) once an activity becomes enabled, there is no other-than-programmatic way to disable it. A milestone can be used to test whether some part of the process is in a given state. Simple message passing mechanisms will not be able to support this because the disabling of a milestone corresponds to withdrawing a message. This type of functionality is typically not offered by existing workflow management systems. Note that in Figure 15 activity *process_complaint* may be executed an arbitrary number of times, i.e. it is possible to bypass *process_complaint*, but it is also possible to execute *process_complaint* several times. It is not possible to model such a construct by an AND-split/AND-join type of synchronization between the two parallel branches, because it is not known how many times a synchronization is needed.

Implementation

- Consider three activities A, B, and C. Activity can be executed an arbitrary number of times before the execution of and after the execution of , cf. Workflow A in Figure 17. Such a milestone can be realized using Pattern 16 (Deferred Choice). After executing there is an implicit XOR-split with two possible subsequent activities: and . If is executed, then the same implicit XOR-split is activated again. If is executed, is disabled by the implicit XOR-split construct. This solution is illustrated by Workflow B in Figure 17. Note that this solution only works if the execution of is not restricted by other parallel threads. For example, the construct cannot be used to deal with the situation modeled in Figure 15 because *process_complaint* can only be executed directly after a positive evaluation or a negative check, i.e. the execution of *process_complaint* is restricted by both parallel threads. Clearly, a choice restricted by multiple parallel threads cannot be handled using Pattern 16 (Deferred Choice).
- Another solution is to use the data perspective, e.g. introduce a Boolean workflow variable . Again consider three activities A, B, and C such that activity is allowed to be executed in-between and . Initially, is set to false. After execution of is set to true, and activity sets to false. Activity is preceded by a loop which periodically checks whether is true: If is true, then is activated and if is false, then check again after a specified period, etc. This solution is illustrated by Workflow C in Figure 17. Note that this way a ‘‘busy wait’’ is introduced and after enabling it cannot be blocked anymore, i.e., the execution of does not influence running or enabled instances of . Using Pattern 19 (Cancel Activity), can be withdrawn once is started. More sophisticated variants of this solution are possible by using database triggers, etc. However, a drawback of this solution approach is that an essential part of the process perspective is hidden inside activities and applications. Moreover, the mixture of parallelism and choice may lead to all kinds of concurrency problems.

TU/e technische universiteit eindhoven

pattern	product							
	Staffware	COSA	InConcert	Eastman	FLOWer	Domino	Meteor	Mobile
1 (seq)	+	+	+	+	+	+	+	+
2 (par-spl)	+	+	+	+	+	+	+	+
3 (synch)	+	+	+	+	+	+	+	+
4 (ex-ch)	+	+	+/-	+	+	+	+	+
5 (simple-m)	+	+	+/-	+	+	+	+	+
6 (m-choice)	-	+	+/-	+/-	-	+	+	+
7 (sync-m)	-	+/-	+	+	-	+	-	-
8 (multi-m)	-	-	-	+	+/-	+/-	+	-
9 (disc)	-	-	-	+	+/-	-	+/-	+
10 (arb-c)	+	+	-	+	-	+	+	-
11 (impl-t)	+	-	+	+	-	+	-	-
12 (mi-no-s)	-	+/-	-	+	+	+/-	+	-
13 (mi-dt)	+	+	+	+	+	+	+	+
14 (mi-rt)	-	-	-	-	+	-	-	-
15 (mi-no)	-	-	-	-	+	-	-	-
16 (def-c)	-	+	-	-	+/-	-	-	-
17 (int-par)	-	+	-	-	+/-	-	-	+
18 (milest)	-	+	-	-	+/-	-	-	-
19 (can-a)	+	+	-	-	+/-	-	-	-
20 (can-c)	-	-	-	-	+/-	+	-	-

basic
adv. synch.
struct.
mult. inst.
state
cancel

TU/e technische universiteit eindhoven

pattern	product						
	MQSeries	Forté	Verve	Vis. WF	Changeng.	I-Flow	SAP/R3
1 (seq)	+	+	+	+	+	+	+
2 (par-spl)	+	+	+	+	+	+	+
3 (synch)	+	+	+	+	+	+	+
4 (ex-ch)	+	+	+	+	+	+	+
5 (simple-m)	+	+	+	+	+	+	+
6 (m-choice)	+	+	+	+	+	+	+
7 (sync-m)	+	-	-	-	-	-	-
8 (multi-m)	-	+	+	-	-	-	-
9 (disc)	-	+	+	-	+	-	+
10 (arb-c)	-	+	+	+/-	+	+	-
11 (impl-t)	+	-	-	-	-	-	-
12 (mi-no-s)	-	+	+	+	-	+	-
13 (mi-dt)	+	+	+	+	+	+	+
14 (mi-rt)	+/-	-	-	-	-	-	+/-
15 (mi-no)	-	-	-	-	-	-	-
16 (def-c)	-	-	-	-	-	-	-
17 (int-par)	-	-	-	-	-	-	-
18 (milest)	-	-	-	-	-	-	-
19 (can-a)	-	-	-	-	-	-	+
20 (can-c)	-	+	+	-	+	-	+

basic
adv. synch.
struct.
mult. inst.
state
cancel

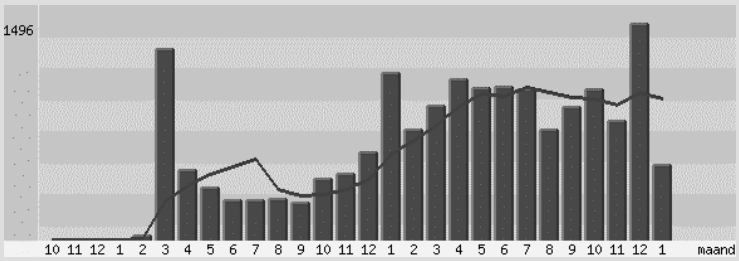
TU/e technische universiteit eindhoven

Scientific results

- Mapping onto WF-nets for analysis etc.
- Classification of mechanisms:
  - single thread (safe) / multiple threads (non-safe)
  - state machine / marked graph / free-choice / well-structured / arbitrary
  - graph structured / block structured
  - normal token / true-false token / maximal input
- Observation: New systems/standards are more expressive!

TU/e technische universiteit eindhoven

Practical impact



- **http://www.tm.tue.nl/it/research/patterns**
- +/- 80 pageviews per working day (>17.000 in total)
- patterns are used in selection processes
- role of vendors has been opportunistic

TU/e technische universiteit eindhoven

Part II: XRL

Joint work with Eric Verbeek (EUT), Akhil Kumar (CU/BL), and Alexander Hirnschall (EUT).

http://www.tm.tue.nl/it/staff/wvdaalst/workflow/xrl

TU/e technische universiteit eindhoven

XRL: Motivation

- The eXchangeable Routing Language (XRL) has been developed to address limitations of existing systems, cf.
  - difficulties supporting complex processes (lack of expressive power)
  - limited run-time flexibility
  - limited support for interorganizational workflows
  - limited support for analysis
- language is inspired by patterns research
- goal: testbed/play yard for scientific results

TU/e technische universiteit eindhoven

Features of XRL

- **Syntax** is **XML** based: Allows for the application of XML technology (XSLT, etc.).
- **Semantics** is based on **Petri nets**: Allows for analysis and enactment.
- **Extendible** with new routing primitives (exploits XML/Petri net base).
- Processes described at **instance** level (allows for run-time flexibility and additional patterns).

TU/e technische universiteit eindhoven

Language: XRL Routing Elements

- Task
- Sequence
- Any_sequence
- Choice
- Condition
- Parallel_sync
- Parallel_no_sync
- Parallel_part_sync
- Parallel_part_sync_cancel
- Wait_all
- Wait_any
- While_do
- Terminate

TU/e technische universiteit eindhoven

XRL – DTD (1)

```
<!ENTITY % routing_element
    "task|sequence|any_sequence|choice|condition|parallel_sync|parallel_no_sync|parall
    el_part_sync|parallel_part_sync_cancel|wait_all|wait_any|while_do|terminate">
<!ELEMENT route ((%routing_element;), event*)>
<!ATTLIST route
    name ID #REQUIRED
    created_by CDATA #IMPLIED
    date CDATA #IMPLIED>
<!ELEMENT task (event*)>
<!ATTLIST task
    name ID #REQUIRED
    address CDATA #REQUIRED
    role CDATA #IMPLIED
    doc_read NMTOKENS #IMPLIED
    doc_update NMTOKENS #IMPLIED
    doc_create NMTOKENS #IMPLIED
    result CDATA #IMPLIED
    status (ready|running|enabled|disabled|aborted|null) #IMPLIED
    start_time NMTOKENS #IMPLIED
    end_time NMTOKENS #IMPLIED
    notify CDATA #IMPLIED>
```

XML (eXtendible Markup Language)
DTD (Document Type Definition)
XSLT (eXtendible Stylesheet Language Transformations)

TU/e technische universiteit eindhoven

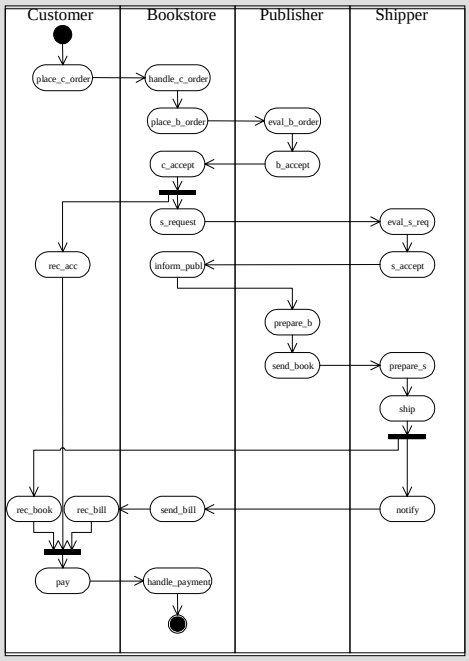
XRL – DTD (2)

```
<!ELEMENT event EMPTY>
<!ATTLIST event
    name ID #REQUIRED>
<!ELEMENT sequence ((%routing_element;|state)+)>
<!ELEMENT any_sequence ((%routing_element;)+)>
<!ELEMENT choice ((%routing_element;)+)>
<!ELEMENT condition ((true|false)*)>
<!ATTLIST condition
    condition CDATA #REQUIRED>
<!ELEMENT true (%routing_element;)>
<!ELEMENT false (%routing_element;)>
<!ELEMENT parallel_sync ((%routing_element;)+)>
<!ELEMENT parallel_no_sync ((%routing_element;)+)>
<!ELEMENT parallel_part_sync ((%routing_element;)+)>
<!ATTLIST parallel_part_sync
    number NMTOKEN #REQUIRED>
<!ELEMENT parallel_part_sync_cancel ((%routing_element;)+)>
<!ATTLIST parallel_part_sync_cancel
    number NMTOKEN #REQUIRED>
<!ELEMENT wait_all ((event_ref|timeout)+)>
<!ELEMENT wait_any ((event_ref|timeout)+)>
<!ELEMENT event_ref EMPTY>
<!ATTLIST event_ref
    name IDREF #REQUIRED>
<!ELEMENT timeout ((%routing_element;)?)>
<!ATTLIST timeout
    time CDATA #REQUIRED
    type (relative|s_relative|absolute) "absolute">
<!ELEMENT while_do (%routing_element;)>
<!ATTLIST while_do
    condition CDATA #REQUIRED>
<!ELEMENT terminate EMPTY>
<!ELEMENT state EMPTY>
```

TU/e technische universiteit eindhoven

Example (1)

- An electronic bookstore
- Activity diagram of order flow (simplification)
- Four parties involved



TU/e technische universiteit eindhoven

Example (2): XRL Conversion

```
<!DOCTYPE route SYSTEM "xrl.dtd">
<route name="e-bookstore" created_by="H.M.W. Verbeek" date="June 11, 2001">
     <sequence>
       <task name="place_c_order" address="customer"/>
       <task name="handle_c_order" address="bookstore"/>
       <while_do condition="No publisher found yet">
         <sequence>
            <task name="place_b_order" address="bookstore"/>
            <task name="eval_b_order" address="publisher"/>
            <condition condition="No publisher found yet">
              <true>
                <sequence>
                   <task name="decide" address="publisher"/>
                   <condition condition="Try alternative publisher">
                     <true>
                        <task name="alt_publ" address="publisher"/>
                     </true>
                     <false>
```

TU/e technische universiteit eindhoven

Example (3): XRL Conversion

```
                <sequence>
                    <task name="b_reject" address="publisher"/>
                    <task name="c_reject" address="bookstore"/>
                    <task name="rec_decl" address="customer"/>
                </sequence>
              </false>
           </condition>
        </sequence>
     </true>
     <false>
        <sequence>
            <task name="b_accept" address="publisher"/>
            <task name="c_accept" address="bookstore"/>
            <parallel_sync>
               <task name="rec_acc" address="customer">
                  <event name="accept"/>
               </task>
               <sequence>
                  <while_do condition="No shipper found yet">
```

TU/e technische universiteit eindhoven

Example (4): XRL Conversion

```
                  <sequence>
                     <task name="s_request" address="bookstore"/>
                     <task name="eval_s_req" address="shipper"/>
                  </sequence>
               </while_do>
                  <condition condition="Shipper found">
                     <true>
                        <sequence>
                           <task name="s_accept" address="shipper"/>
                           <task name="inform_publ" address="bookstore"/>
                           <task name="prepare_b" address="publisher"/>
                           <task name="send_book" address="publisher"/>
                           <task name="prepare_s" address="shipper"/>
                           <task name="ship" address="shipper"/>
                           <parallel_sync>
                              <sequence>
                                 <task name="notify" address="shipper"/>
                                 <task name="send_bill" address="bookstore"/>
```

TU/e technische universiteit eindhoven

Example (5): XRL Conversion

```
                    <wait_all>
                       <event_ref name="accept"/>
                    </wait_all>
                    <task name="rec_bill" address="customer"/>
                 </sequence>
                 <sequence>
                    <wait_all>
                     <event_ref name="accept"/>
                    </wait_all>
                    <task name="rec_book" address="customer"/>
                 </sequence>
              </parallel_sync>
              <task name="pay" address="customer"/>
              <task name="handle_payment" address="bookstore"/>
           </sequence>
        </true>
      <false>
        <task name="s_reject" address="shipper"/>
      </false>
```

TU/e technische universiteit eindhoven

Example (6): XRL Conversion

```
                                  </condition>
                               </sequence>
                            </parallel_sync>
                         </sequence>
                      </false>
                   </condition>
                </sequence>
             </while_do>
          </sequence>
       </route>
```

TU/e technische universiteit eindhoven

Also a DTD for organizational matters

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT organization (resources?,
   resource_types?, collections?,
   relations?)>
<!ELEMENT resources (user*, machine*,
   space*)>
<!ELEMENT user EMPTY>
<!ATTLIST user
   user_id IDREF #REQUIRED
   first_name CDATA #IMPLIED
   last_name CDATA #IMPLIED
   department IDREF #IMPLIED
   e-mail CDATA #IMPLIED
   login_id CDATA #IMPLIED
   address CDATA #IMPLIED
   phone CDATA #IMPLIED
   skills CDATA #IMPLIED
>
<!ELEMENT machine EMPTY>
<!ATTLIST machine
   machine_id IDREF #REQUIRED
   machine_name CDATA #IMPLIED
   description CDATA #IMPLIED
   number CDATA #IMPLIED
...
```

```
<!ELEMENT resource_types (role*,
   machine_type*, space_type*)>
<!ELEMENT role EMPTY>
<!ATTLIST role
   role_id IDREF #REQUIRED
   name CDATA #IMPLIED
   description CDATA #IMPLIED
>
<!ELEMENT can_inherit (role_ref,
   role_ref)>
<!ATTLIST can_inherit
   transitive_flag CDATA #IMPLIED
   restrictions CDATA #IMPLIED
>
<!ELEMENT can_delegate ((role_ref,
   role_ref) | (user_ref, user_ref) |
   (user_ref, role_ref) | (role_ref,
   user_ref))>
<!ATTLIST can_delegate
   transitive_flag CDATA #IMPLIED
   restrictions CDATA #IMPLIED
>
<!ELEMENT availability (user_ref*,
   machine_ref*, space_ref*)>
...
```

TU/e technische universiteit eindhoven

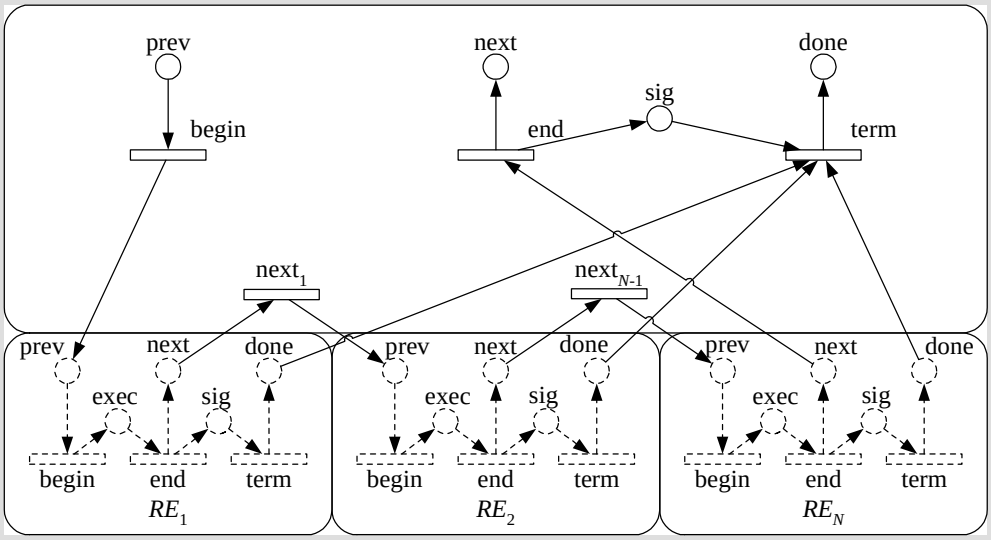
Petri-net Semantics

- DTD describes syntax but does not specify semantics
- Transformation to Petri net allows for analysis and enactment
- XRL document forms a tree
  - route element as root
  - child routing elements interface with parent elements
- Three examples: sequence, any_sequence, and parallel_sync

TU/e technische universiteit eindhoven

Sequence

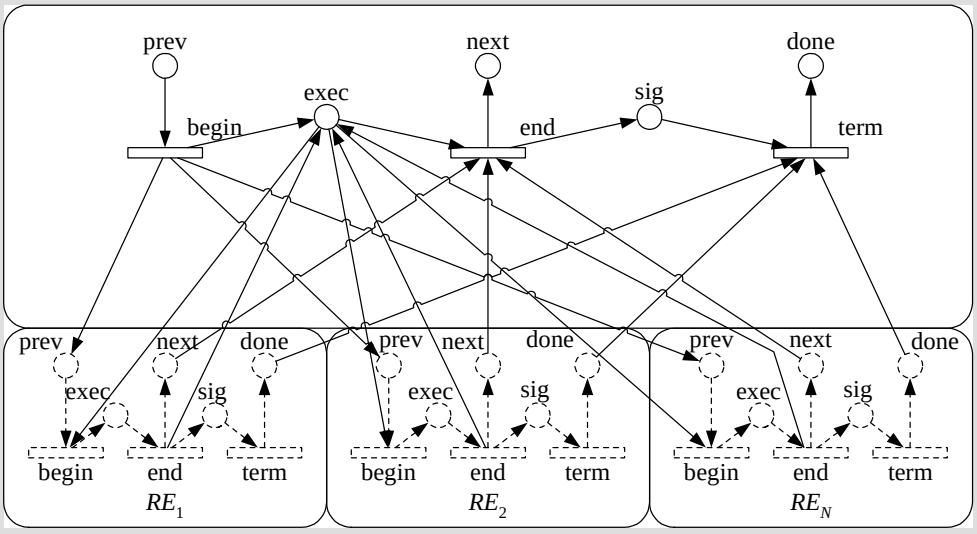
<!ELEMENT sequence ((%routing_element;)+)>



TU/e technische universiteit eindhoven

Any_sequence

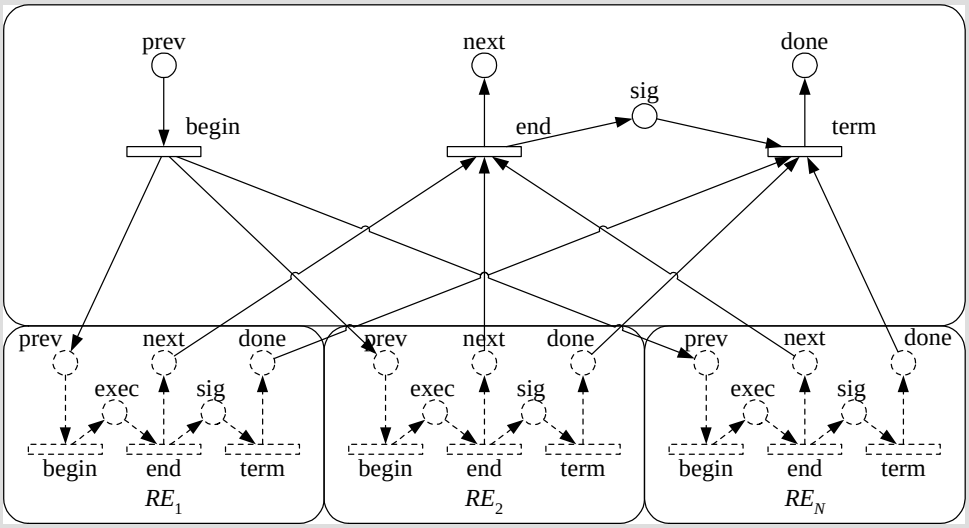
<!ELEMENT any_sequence ((%routing_element;)+)>



TU/e technische universiteit eindhoven

Parallel_sync

<!ELEMENT parallel_sync ((%routing_element;)+)>



TU/e technische universiteit eindhoven

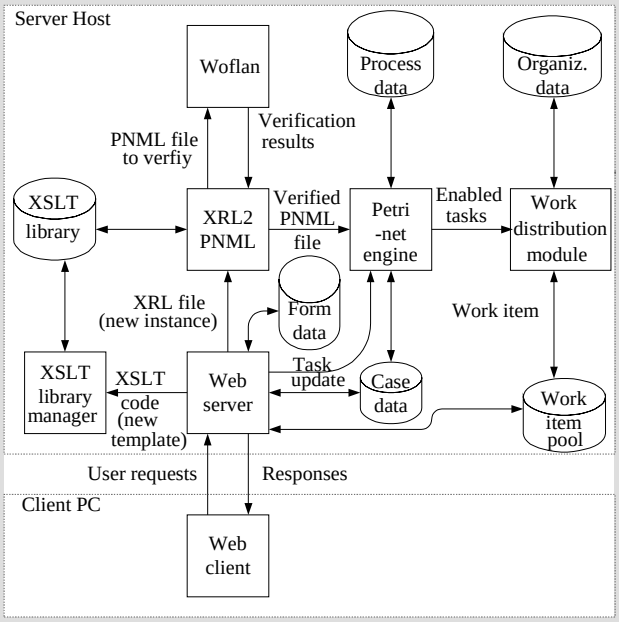
Benefits of Petri-net Semantics

- Analysis of correctness is possible
  - e.g. Woflan
- Efficient implementation of workflow engine
- Extendibility
  - DTD definition extension of XRL
  - XSLT supported translation to PNML
  - No changes of Petri-net engine required

TU/e technische universiteit eindhoven

Architecture

- Inter-organizational workflow
- System builds on Petri-net kernel and PNML
- Toolset involved
  - XRL/Flower
  - XRL/Woflan



TU/e technische universiteit eindhoven

Conclusion

- Expressive power of contemporary WFM systems is limited.
- Objective evaluation through workflow patterns is possible.
- Patterns can also be used to train designers and structure workarounds.
- New languages such as XRL can learn from these patterns.
- Features of XRL:
  - Syntax is XML based: Allows for the application of XML technology (XSLT, etc.).
  - Semantics is based on Petri nets: Allows for analysis and enactment.
  - Extensible with new routing primitives (exploits XML/Petri net base).
  - Processes described at instance level (allows for run-time flexibility and additional patterns).

TU/e technische universiteit eindhoven

How about the standards in this domain?

pattern	*standard*						
	XPDL	UML	BPEL	XLANG	WSFL	BPML	WSCI
Sequence	+	+	+	+	+	+	+
Parallel Split	+	+	+	+	+	+	+
Synchronization	+	+	+	+	+	+	+
Exclusive Choice	+	+	+	+	+	+	+
Simple Merge	+	+	+	+	+	+	+
Multi Choice	+	-	+	-	+	-	-
Synchronizing Merge	-	-	+	-	+	-	-
Multi Merge	-	-	-	-	-	+/-	+/-
Discriminator	-	-	-	-	-	-	-
Arbitrary Cycles	+	-	-	-	-	-	-
Implicit Termination	+	-	+	-	+	+	+
MI without Synchronization	-	-	+	+	+	+	+
MI with a Priori Design Time Knowledge	+	+	+	+	+	+	+
MI with a Priori Runtime Knowledge	-	+	-	-	-	-	-
MI without a Priori Runtime Knowledge	-	-	-	-	-	-	-
Deferred Choice	-	+	+	+	-	+	+
Interleaved Parallel Routing	-	-	+/-	-	-	-	-
Milestone	-	-	-	-	-	-	-
Cancel Activity	-	+	+	+	+	+	+
Cancel Case	-	+	+	+	+	+	+