Workflow Management Systems
Workflow management is the automated coordination, control and communication of work, both of people and computers, in the context of organizational processes, through the execution of software in a network of computers whose order of execution is controlled by a computerized representation of the business processes.

Workflow management system: “a system that defines, creates and manages the execution of workflows through the use of software, running on one or more workflow engines, which is able to interpret the process definition, interact with workflow participants and, where required, invoke IT tools and applications” [WFM Coalition]
ANATOMY OF WORKFLOW MANAGEMENT

Business Process
- Is defined in a
- Process Definition
  - (a representation of what is intended to happen)
  - Composed of
    - Sub-Processes
    - Activities
      - Which may be
        - Manual Activities
        - Automated Activities
          - (which are not managed as part of the Workflow system)
    - Process Instances
      - (a representation of what is actually happening)
      - Include one or more
        - Activity Instances
          - Which include
            - Work Items
              - (tasks allocated to a workflow participant)
            - Invoked Applications
              - (computer tools/applications used to support an activity)
- Invoked Applications
  - (controls automated aspects of the business process via)
- Process Instances
  - (a representation of what is actually happening)
- Activity Instances
  - Which include
    - Work Items
    - Invoked Applications
  - During execution are represented by
    - Process Instances
    - Activity Instances
- Business Process
  - Is managed by
  - Workflow Management System
    - (controls automated aspects of the business process via)
WFMS FEATURES

- Monitoring, tracking, auditing, reporting
- Authorization; Security
- Interoperability
- Multiple computing platforms and communications infrastructures
- Load balancing
- Versioning and life-cycle
- Scalability: Partly distributed enactment service (multiple server support); Fully distributed enactment service
- Cloud support
PROCESS ANALYSIS

- **Purpose**: To ensure that the right people understand the necessary facts about an organizational process.

- **Objectives**:
  - shared understanding
  - trigger model (event-driven)

- **How to**:
  - Who to talk to? (roles)
  - What do you do? (activities)
  - What prompts you to do it? (triggers)
  - Follow links, and repeat.

- **Using**: interviews (time consuming, rich secondary information), meetings (quick, obscure social process)
EXAMPLE: COMPLAINT PROCEDURE

- customer
- representative
- inspector
- manager
- librarian

- deliver complaint
- reject
- filling a complaint
- negotiate solution
- negotiate satisfaction
- consent
- summarize
- analyze
- approve
- execute solution
- log
- ackn.
- ackn.
- Workflow (process) class (schema) to model a(n) (business / organizational) process
- Task, activity or step
- Task coordination / linking or Control flow (serial / parallel-resync/list/queues/network, rules/triggers, dependencies / conditions)
- Data flow or sharing (explicit passing, shared data, common variables)
- Processing entities: Users-roles and authorization, worklists; Information Systems
WORKFLOW MODELLING

- Workflows deal with (parts of) business processes, also called **cases** or **scenarios**
  - e.g., an insurance claim, a loan application
- Similar cases belong to the same case **type**
- Each case has a unique **identity**
- Cases have a limited **lifetime**: from the point in time the case was submitted, to the point its processing has been completed
- At any point during its lifetime, a case has a **state** comprising:
  - the values of the relevant **attributes**
  - the **conditions** that have been fulfilled
  - the **content** of the case
Workflows are structured in\textit{ tasks}:

- a task is a logical \textit{unit} of work
- regarded as \textit{indivisible} or \textit{atomic} (either executed in full or if its execution is stopped, a rollback to the previous state takes place)

Tasks are distinguished into:

- \textit{manual}: performed by humans without IT support
- \textit{automatic}: performed without human intervention
- \textit{semi-automatic}: involve both humans and application programs
APPLICATION/AUTOMATIC TASKS

- **Application tasks** involve
  - scripts for terminal emulation to remote systems
  - Web services
  - application programs/systems providing data manipulation (filters)
  - predefined interfaces to legacy application systems
  - stored procedure calls
  - client programs or servers invoking other servers
  - database transactions
Tasks refer to generic pieces of work and not to performing a particular activity for a specific case.

- **Task** \(\approx\) work item
- **Activity** \(\approx\) task execution for particular case

A **process** specifies the way in which a particular category of cases should be carried out and in what order. A process can be viewed as a **procedure** for a particular case type.

- Different cases can be handled using a single process
- A process is made up of tasks and conditions and may include **sub-processes**
- Complex processes can be structured **hierarchically**
- Each process has a **beginning** and an **end** which mark the **initiation** and **completion** of a case.
Workflow scheduling or routing: basic constructs

- **sequential task execution**: implies a dependency between tasks that follow one another; the result of one task is input to the next.

- **parallel task execution**: tasks that need to be executed simultaneously without the result of one affecting the other. Parallel tasks are initiated using an AND-split and later resynchronized using an AND-join.

- **selective routing**: choice between two or more tasks (depending upon specific properties of the case). Choice also known as an OR-split. Alternative paths are reunited using an OR-join.

- **iteration**: performing a particular task a number of times.
TYPES OF PROCESSING ENTITIES

- humans (may appear as a GUI; may use document/image processing systems and applications)
- script interpreters and compilers (for processing scripts and application programs)
- (legacy) application systems
- servers in client-server and transaction processing systems
- DBMSs
ADDITIONAL MODELLING FEATURES

- Tasks: non-transactional, transactional
- Execution environment / infrastructure / configuration: execution location, interfaces
- Deadlines
- Exception Handling (Error Handling, Recovery) specification
Workflow enactment

- the enactment of work items is triggered by resources (human or automated), an external event or a time signal

Workflow modelling and analysis must be carried out formally

- forces precise definitions avoiding ambiguities, uncertainties and contradictions (contrary to many semi-formal diagrammatic techniques)

- formalisms can be used to reason about processes (e.g., whether a case is successfully completed after a period of time, whether “liveness” or “safety” properties are maintained etc.)
“A clear theoretical basis and correctness criteria must be established which enable the runtime system to efficiently reason about the correctness of a requested change ...”

Types of Analysis

- **Validation** - interactive simulation (Are we building the right product?)
- **Verification** (establishing correctness of a workflow) - advanced analysis techniques (Are we building the product right?)
- **Performance Analysis** - throughput etc.
CORRECTNESS CRITERIA

- **Structural properties**
  - Control flow, Data flow, Temporal constraints
  - Reachability, Termination, Deadlocks, Data inconsistency, Missing input data

- **Other Workflow characteristics**
  - Reassignment of task to agents
  - Changes in organizational schema
  - Access to external databases
High level Petri nets
State and Activity charts
Temporal logic
Process Algebra
Graph based models
Rules

But there are limitations wrt to what is modeled using formal models. Often limited to workflow maps/graphs, inter-task dependencies.
PETRI NETS

- a formalism for modeling and analyzing workflows
  - devised by Carl Petri (1962) as a tool for process modeling and analysis
- processes can be described graphically and in addition they possess a mathematical foundation
- A Petri Net (PN) consists of places and transitions denoted by circles and rectangles respectively
  - E.g., the claim processing PN may have the places claim, under consideration and ready and the transitions record, pay and send letter
- Places symbolize states, conditions, or resources that need to be met/be available before an action can be carried out. Transitions symbolize actions
- places and transitions are connected by directed edges (place → transition, transition → place, but not between places or transitions)
PETRI NETS

- A place $p$ is called an input place for a transition $t$ if there is an edge from $p$ to $t$.

- A place $p$ is called an output place for a transition $t$ if there is an edge from $t$ to $p$.

- Places may contain tokens (denoted by black dots).

- Although the structure of a PN is fixed, the distribution of tokens among the places may change.

- The state of a PN is indicated by the distribution of tokens in the places of the net.

- Firing a transition results in tokens moving from input places to output places.
The state may be represented as the vector \((3,0,0)\)
A transition may fire if it is enabled, i.e., there exists at least one token in each of its input places.

When a transition fires, one token is removed from each input place and one token is added to the output place.

Tokens are consumed from input places and produced at output places.
After transition **record** fires

- **Claim**
- **Record**
- **Under consideration**
- **Pay**
- **Send letter**
- **Ready**

---

After transition **pay** fires

- **Claim**
- **Record**
- **Under consideration**
- **Pay**
- **Send letter**
- **Ready**
When a transition fires, the process shifts from one state to another.

Transitions represent event occurrences, operations, transformations etc. and are the active components of a P.N.

Places are passive: they cannot change the state of the net; they represent particular conditions.

Tokens represent objects (physical, informational etc.).

More than one case can be in progress simultaneously in a PN.

A PN can also be used to describe repetitive processes.
- **Example**: a traffic light process for crossing a street

```
red  yr
yellow  rg
green  gy
```
PETRI NETS

- **Example**: two traffic light processes for crossing two 1-way streets
  - **Requirement**: one of the two must always be red
Apart from the graphical notation provided by PNs, one must be able to determine that the process modeled will operate safely.

- we will study analysis techniques later in the course

PNs may become too large and difficult to use. Several extensions have been proposed:

- Colored Petri Nets
- Temporal Petri Nets
- Hierarchical Petri Nets

... collectively referred to as high-level Petri Nets
Their primary aim is to represent the dynamic aspects of system behavior and because of this they do not have anything other than very simple capacity to represent entities of the domain of application.

- Data representation is limited to tokens which are indistinguishable from each other.
- Clearly this is inadequate for representing IT systems.

Source: http://www.dit.ie/computing/
Colored Petri Nets combine the strength of Petri nets with the strength of programming languages.

Petri nets provide the primitives for the description of the synchronisation of concurrent processes, while programming languages provide the primitives for the definition of data types and the manipulation of data values.
COLORED PETRI NETS

- In classic PN tokens found in the same place are by definition indistinguishable.
- There is a need to represent different aspects or attributes of objects.
- In colored PN each token has a value or color.
- Transitions that fire produce tokens whose values depend on the values of the tokens consumed.
- The number of tokens produced by a transition also depends on the values of the tokens consumed.
- This provides a greater flexibility in representing different cases of processes.
- Like traditional Petri Nets Colored Petri Nets consist of Places, Transitions, connected by Arcs (forming a bi-partite graph).
- They are a combination of text and graphics.
Example: Dealing with technical faults in a product department

- faults are categorized
- if they cannot be corrected right away, a repair takes place and it is tested yielding three possible results:
  1. Fault has been corrected
  2. Further repair is required
  3. Faulty component must be replaced

A token value represents relevant properties of the fault that needs to be dealt with and this value is retained throughout the tokens’ trajectory in the net.
● An output place may not receive a token: depending on the transition *categorize*, a fault will be either solved or a repair will be needed.

● Transition *test* may produce a token in one of 3 possible places.
In colored PN, **conditions** can be set for the values of tokens to be consumed.

A transition is only enabled if there is a token at each of its input places and the **preconditions** for firing the transition are met.

Preconditions are logical expressions referring to the values of tokens.

- E.g., the transition **categorize** may have the precondition **“the value of the token to be consumed from the place fault must contain a valid component id”**.

- In this case, a fault missing a valid id will not be categorized (transition not fired; remain in the place fault).
Preconditions can also be used to synchronize tokens, i.e., to specify that a transition will fire only if a particular combination of tokens can be consumed.

**Example:** car assembly process

- when a transition fires the number of input tokens must be equal to the number of incoming arrows
- a precondition must specify that tokens can only be consumed in a certain combination (and not at random)
The examples show that not all the required information can be represented graphically.

For each transition, we need to specify:

- the **precondition** for firing the transition (if one exists)
- the **number of tokens produced** in each output place when the transition fires
- the **values of the tokens produced**

This information can be specified as **text** (pseudo-code) or as a **subroutine** in a programming language or as a **formal specification**.
PLACES

- **Places are specified with** the following inscriptions:
  - *Name* (for identification).
  - *Color set* (specifying the type of tokens which may reside on the place).
  - *Initial marking* (multi-set of token colors)
Each \textit{transition} has the following inscriptions:

- \textit{Name} (for identification).
- \textit{Guard} (boolean expression containing some of the variables).
Each arc has the following inscriptions:

- *Arc expression* (containing some of the variables).

- When the arc expression is evaluated it yields a multi-set of token colors.
- Place P2 is empty.
- The marking at P1 consists of 2 tokens of type integer whose value is 3 and 2 tokens of type integer whose value is 8.
- On T1 is the guard $X > 5$. This is a barrier to T1 happening, in that T1 will only pass if the assignment to $X$ under the occurrence in question is greater than 5.
- As we will see these guards on transitions will play an important part in representing the barriers to the performance of activities.
On the arc between P1 and T1 is the atomic expression consisting of a variable X to which may be bound one of the input tokens in P1. Such a binding is called an occurrence.

The expression on the output arc from T1 to P2 represents the state change which takes place across transition T1. In this example it embodies an increment of 1 on the variable X.
GUARDS ON THE EXPRESSION

- The expression $X > 5$ at transition $T_1$ is known as a guard and must be satisfied by the incoming token values.
- If the incoming tokens do not satisfy the guard then the transition cannot happen.
- In the case $X$ has value $8$ which is $> 5$, the transition can take place.
The passage of tokens across transitions from place to place through this process of binding, satisfying guards and modifying data is called an occurrence.

The occurrence will be blocked if the guard is not satisfied by the incoming tokens.

So if an input token with value 5 was bound to X then the transition could not occur.

The guard is a barrier to the transition happening.
Often, there is a need to specify expected process completion time and other information related to the timing of transitions in a PN.

Classic PN do not allow modeling time.

In temporally-extended PN, tokens have a timestamp in addition to a value. The timestamp represents the time from which the token is available for consumption.

Transition enabling time: earliest moment at which all its input places contain sufficient available tokens.

Tokens are consumed in a first-in-first-out fashion.

The transition with the earliest enabling time fires first. If more than one have the same enabling time, one is chosen randomly.
Firing of a transition may affect the enabling time of other transitions

Produced tokens have timestamp equal to or greater than the transition firing time

There may be a delay that depends on the transition and the values of the tokens consumed

The delay may be constant (could be 0) or may be decided randomly

Transition firing is taken to be instantaneous
**Example:** synchronized traffic lights

If `rg1` is chosen a token will be produced at `green1` with delay 25, at `yellow1` with delay 30 and at `red1` and `X` with delay 60; `rg2` will fire.
HIERARCHICAL PETRI NETS

- Classic, colored or temporally-extended PN have a flat structure resulting in a single, possibly extensive and complex net.
- Often processes are structured hierarchically, and can be modeled by networks containing sub-networks (sub-processes)
- This provides the ability to refine processes rather than regard them as non-decomposable.
- Graphically, non-atomic sub-processes are represented as doubly-arrowed rectangles.
HIERARCHICAL PETRI NETS

fault → categorize → solved

reeds → repair → reeds

repair → testing → test

test → replace

start → ntr → trace → nch → change → ne → end

free
HIERARCHICAL PETRI NETS

- PNs can be structured hierarchically using a bottom-up or top-down approach
  - **Top-down decomposition** starts with high-level processes that are gradually decomposed into sub-processes; at the lowest level, processes consist only of transitions and places
  - **Bottom-up synthesis** starts with elementary components that are combined into larger processes
- Complex processes are rarely non-hierarchical, hence a **divide and conquer** strategy is needed.
- Reuse of (sub-) processes is also possible (e.g., when a recurring process is included)
REFERENCES

- “Workflow Management: Models, methods and systems” by van der Aalst and van Hee
- Workflow Management Coalition (http://www.wfmc.org/ )
- Coloured Petri Nets (http://www.cs.au.dk/~cpnbook/ )
- Tutorial: http://www.informatik.uni-hamburg.de/TGI/PetriNets/introductions/pn2000_introtut.pdf )
- Animated examples: https://www.informatik.uni-hamburg.de/TGI/PetriNets/introductions/aalst/
- https://www.youtube.com/watch?v=3KjKY8k9Lk