

Survey of Workflow Management Systems

Armin Haller Eyal Oren Simeon Petkov

May 2, 2005

1 Introduction

Workflow management deals with supporting business processes in organisations, it involves managing the flow of work through an organisation [1]. Workflows are a collection of coordinated tasks designed to carry out a well-defined complex process [11]. A workflow management system is a generic information system that supports modelling, execution, management and monitoring of workflows.

Many different workflow management systems have been developed that focus on different application domains and provide different functionality, c.f. [2, 5, 8]. Workflow management lacks a standardised theory that provides a theoretical background for workflows like the relational algebra provides for databases [1]; despite efforts of standardisation bodies there is no consensus on the representation or conceptual model of workflow processes [14].

In this diverse and complicated landscape it is a challenge to evaluate and compare the functionality of various workflow management systems and to survey the functional requirements of workflow management systems. A number of approaches attempt to address this situation: Jablonski and Bussler [6] describe a number of essential perspectives and aspects of a comprehensive workflow management functionality; they provide a structure in the complex environment of workflows. Van der Aalst *et al.* [2] and Russell *et al.* [13] systematically analyse available functionality in existing workflow management systems, and categorise these in a number of *workflow patterns*; these patterns are devoid of implementation issues and form a qualitative standard against which existing workflow management systems can be compared.

In Petkov *et al.* [12] we describe an integrated view on these aspects: we indicate what comprises each aspect, how each aspect can be modelled, and what quality indicators exist to assess its support in workflow management systems. In this paper we build upon this earlier work to evaluate the functionality of workflow management systems per aspect.

2 Methodology

Software evaluation can be carried out in a multitude of ways. A common distinction is made between quantitative and qualitative software evaluations [7]. A quantitative evaluation identifies the benefits one expects a software tool to deliver in measurable terms; it involves the collection of data that determine whether the expected results are delivered or not. Kitchenham [7] identifies three different ways of organising a quantitative evaluation exercise.

Formal experiment: A formal experiment is a rigorous, controlled investigation of an activity, where key factors are identified and manipulated to document their effects on the outcome. Since formal experiments require a great deal of control, they tend to be small, involving small numbers of people or events [3].

Case Study: A case study is a research technique where you identify key factors that may affect the outcome of an activity imposed by a real project using the standard project development procedures of the evaluating organisation and then document the activity: its inputs, constraints, resources, and outputs. Case studies usually look at a typical project, rather than trying to capture information about all possible cases [3].

Survey: A survey is a retrospective study of a situation to try to document relationships and outcomes. A survey is always done after an event has occurred. A survey is a retrospective study, you can only record a situation and compare it with similar ones, but you have no control over the situation at hand. You cannot manipulate variables as you do with case studies and experiments. Surveys try to poll what is happening broadly over large groups of projects [3].

Qualitative methods on the other hand assess the extent to which software tools provide the required functionality on a more personal opinion. Kitchenham [7] uses the term feature analysis to describe this kind of software evaluation. Feature analysis requires a subjective assessment of the relative importance of different features. This kind of analysis can be found in different types of weekly or monthly magazines when assessing hardware, cars, bicycles etc. Pure qualitative evaluation exercises are often seen as fuzzy.

This deliverable categorises common functionality of current workflow management systems, to identify the scope that the m3pe system has to cover. Therefore our software evaluation is a hybrid approach, that has both a quantitative and a qualitative nature.

We first carry out a feature analysis based on documentation of the assessed software tools and first-hand experience of the systems. We attempt to capture the whole functionality of systems, and therefore do not focus on a specific application domain. By iteratively comparing the functionality offered by one system with functionality offered by another system we can identify functionalities common to all of them.

We then evaluate the various systems in a more quantitative analysis, using the pattern identified by van der Aalst *et al.* [2] and Russell *et al.* [13]. We show for each pattern whether it is covered by the workflow management system, or whether a workaround exists. That is an extension of the original evaluations (in [2, 13]) since we explicitly mention how the patterns are supported in the investigated systems.

Note: due to severe technical problems installing the various workflow management systems, we were until now only able to evaluate two systems: Fujitsu-Siemens i-Flow and IBM MQ Workflow Workflow. We have tried to evaluate other systems but we did not succeed: we could not get an evaluation version, we could not get the systems installed, documentation was completely lacking, or the systems had severe bugs in their performance. This is obviously an unsatisfying situation, since the survey should include more systems. We will therefore continue this survey and publish an updated report in the future.

3 Survey Scope

Workflow management systems (or business process management systems) enable the automated coordination of activities, data and resources. This coordination is formalised in a workflow model specified at build-time. During process enactment instances of the workflow model are created and executed [4]. The architecture of workflow management systems is separated in a development environment (build time component) and an execution environment (run time component) [10]; we survey both components.

The first available workflow management systems were developed in the late 1980s [9, 10]. Only few of this first generation systems are left in the market. Currently available workflow management systems are offered as a standalone product or are positioned as part of a process-enabled application system (for example in Enterprise Resource Planning systems or in Enterprise Application Integration systems).

Petkov *et al.* [12] explains that a separation of views is necessary to analyse different workflow management systems. In this survey we focus on the functional, behavioural, operational, informational and organisational aspects; we only sketch additional aspects briefly. In order to keep this article self contained we briefly describe these five aspects:

The *functional* aspect describes what should be done in the workflow: it gives a functional decomposition of activities in the workflow.

The *behavioural* aspect describes when something should be done: it gives the execution order and dependencies (control flow) of activities in the workflow.

The *informational* aspect describes the data used in the workflow and the data dependencies between activities. In a workflow management system we can distinguish internal data, which is managed by the workflow management system and external data, which exist even without workflow and might be used for it (e.g. an insurance claim). Activities can depend on data from other activities; the workflow management system can usually transport such

data from one activity to another, which is described in the data-flow between activities.

The *organisational* aspect describes who should do the work in the workflow: it describes constraints on the resource allocation to activities. Activities usually require resources to execute; such resources can be human employees, but also for example computing power or a meeting room. The organisational aspect describes the resources in the organisation, the hierarchy that exists between these resources, and the policies for assigning resources to activities.

The *operational* aspect describes how the workflow management system should interact with its environment: it describes the methods for accessing or invoking external applications (e.g. interaction mode, invocation mode, parameters) and how to communicate with human users.

4 Qualitative Analysis

4.1 i-Flow

i-Flow Interstage Business platform is a customisable Business Process Management engine for web-centric applications. It can be seen as a successor of the workflow engine from the same company Fujitsu Regatta. Our evaluation is based on version 6.1 of the product, introduced in early 2003.

The overall system is laid out as a 4-tier architecture (repository, process logic, web tier, user interface). Two user interface are offered: one can either use the browser-based user interface or a local user interface developed in Java; one can also develop a custom user interface that addresses the provided i-Flow Java APIs. The web tier is composed of servlet components running in a web server. The main process logic resides in the business process management tier. The fourth tier contains the underlying repositories (database, directory, and document management) as well as connectivity to other systems.

The main tool within the Interstage suite is the Development Manager Client. It is used to define workflow types (so called workflow templates in i-Flow) and to manage the execution of workflow instances. An i-Flow process is represented by a flowchart, which contains tasks, their sequential ordering, who performs tasks and the documents required to complete the process.

The Administration Client has the responsibility for administering processes and templates, and the Task Manager Interface that is used for the control and management of worklists and for process generation.

4.1.1 Functional aspect

In i-Flow workflow templates consist of activities, defining the tasks to be performed in the process and the flow control, the operators, which control the order and timing for activities. Furthermore every workflow template must have one start node and at least one end node.

Workflow templates can be organised in super and respective sub-workflows. Sub-processes allow to delegate a task to other workflows. This can be done

from within a work item at runtime by the role associated with the task by selecting another template and mapping data from the original process to the subprocess. The original assigned work item enters a suspended state as soon as the subprocess is invoked. After completion of the sub-process, the work item becomes an accepted activity until it is completed. Sub-processes can also be implemented during design time which are used to break complex tasks into a hierarchy of easier-to-handle units.

4.1.2 Behavioural aspect

i-Flow provides a graphical user interface based on a flowchart notation to create process templates whereas the following constructs are available for designing the sequential behaviour.

Arrow: They are used to link two activities to denote the flow of events.

Conditional Node: This node can be used as an exclusive OR or an AND choice depending on the conditions associated with the node. A default outgoing arrow must be chosen to allow process completion even if conditions are not satisfied. The order in which the conditions are evaluated can be arranged. The expressions used to define conditions are standard JavaScript expressions.

Complex Conditional Node: The behaviour of this node is similar to the Conditional Node with the only exception that the expressions to be used to define conditions can include multiple incoming data items.

OR Node: When this node is used as a split construct it behaves as a parallel split that executes all subsequent activities simultaneously or in any order. When used as a join construct the OR node merges two or more branches without synchronisation: the subsequent activity is started as soon as one branch is finished, but when the subsequent activity finishes all active branches are terminated.

AND Node: The behaviour of the AND node used as a split construct is identical to the described behaviour of the OR node in the same context. When applied as a join construct it is synchronising: the process waits until all activities that lead to this node are completed, before starting the subsequent activity.

Subprocess Node: When the workflow reaches this node, control of the process is passed to a subprocess, and the node enters a Waiting-for-Subprocess state. When the subprocess completes, control returns to this node, and all ensuing activities connected to the node are activated simultaneously. Both run-time and design-time subprocesses are used to break complex tasks into a hierarchy of easier-to-handle units.

Remote Subprocess Node: This node corresponds to the subprocess node with the only difference that the subprocess reside in another workflow engine. i-Flow supports other i-Flow servers, Collaboration Ring or other SWAP compatible process engines. So that it is possible to start a remote subprocess from the local i-Flow Server. The local process waits for the remote process to complete, and incorporates the results of the remote process back into the local process.

Chained Process Node: When the workflow reaches this node, another process is activated similarly to a subprocess. However, the node does not enter a suspended state, and the chained process operates as an independent entity.

Delay Node: When the workflow reaches this node, the process pauses until the first timer attached to the node fires.

Triggers: Triggers either start processes or make choices on activities in response to Data Event (XML) files being added to a particular directory by a data source external to i-Flow.

Exit Node: Identifies the end of the process. Every process must have at least one Exit Node.

4.1.3 Informational aspect

i-Flow follows the principle of globally shared data where no explicit data passing is required [13]. All data items defined within a workflow type are shared between the tasks. Only between super- and sub-workflows a distinct data mapping has to be defined to pass data elements back and forth.

External data passed to i-Flow in the prologue or epilogue actions of any node are always mapped to so called User-defined process attributes (UDAs). UDAs are variables, which hold values in a running process, set by a user through the use of a form, script, or a JavaAction. UDAs have an attribute name, a data type and an initial value. Supported data types are bigdecimal, boolean, date, float, integer, long and string. UDA Assignment Java Actions allow users to assign the value of one User-defined attribute (UDA) to another. Triggers allow to map external data from XML files to UDAs. By loading a XML schema file mappings can be defined from each element to defined UDAs for the activity.

4.1.4 Operational aspect

i-Flow provides communication with external applications over Java Actions. A Java Action is a call to a static method on a specified Java class. The Java action is configured to pass UDA values as parameters, and the result of the method is assigned to a UDA. A Java Action can be invoked when a process starts, when a node is activated or completed, when a timer expires, at time of role resolution and when the process ends. Many nodes (see table 1) have the ability to specify a list of actions to be performed when activated. It is

distinguished between Prologue Actions performed in a node before all its other assigned activities are executed and epilogue Java Actions performed as closure actions in a node. Java Actions allow to execute Java business methods that are outside the scope of i-Flow. Using Generic Java Actions, it is possible to use methods within autonomous Java classes. A Java Action can pass an Enactment context to the invoked application which constitutes an API that can be used to retrieve information out of the process instance.

Node	Prologue	Epilogue
OR Node		yes
AND Node		yes
Condition Node	yes	
Start Node		
Stop Node		
Chained Subprocess	yes	yes
Activity	yes	yes
Voting Activity	yes	yes
Subprocess	yes	yes
Remote Subprocess	yes	yes
Delay	yes	yes

Table 1: List of nodes and their respective support for Java Actions

Action Sets can also execute rules. Rules have to be defined in a rules file created with a supported rules engine. The rules engine must have access to the i-Flow Business Object Model used by i-Flow to implement the rules created by the rules engine.

i-Flow supports a number of pre-defined Java actions, e.g. notification actions, task reassignment actions or SQL Java action. The latter can be applied to feed UDA values by a SQL call. The SQL Java Action is used to query a database associated with applications external to i-Flow and assign the results of the query as the value of a user-defined attribute (UDA). Standard SQL queries are used.

4.1.5 Organisational aspect

At design time template owners have the option of assigning process ownership. Ownership can be assigned to any i-Flow role or set of users.

Activities are assigned to a role, a user, or a set of users. If more than one user is assigned, each assignee receives a separate activity in the Activity List. Every user has its own profile, defining the way the user is notified when a work-item is assigned and the association of a DMS directory for its user ID. The reassignment Java action allows to reassign tasks. A task can either be manually reassigned or automatically to a user or users with the predefined "Assign Task to User Action". A work-item owner can also reassign the task to any other role, whereas the new owner can accept or reject the reassignment.

The directory adapter implements an i-Flow specific interface to expand a group into a list of individuals. The enactment engine uses this at runtime to determine work assignments. The ability to read the directory or to expand groups into a list of individuals is exposed by the Model API. i-Flow provides adaptors to standard LDAP directory such as Sun ONE Directory Server, to Microsoft Active Directory and to Microsoft Windows native user/group support.

4.1.6 Further aspects

In addition to the above mentioned aspects we evaluate as the basic functionality a workflow system has to offer, i-Flow provides further functionality we briefly sketch in this section.

History aspect: The process history in i-Flow keeps an audit trail containing information what happened during the process execution: Date and time of each step in the life of the process, ID, activity name, event associated with activities, responsible persons and errors.

i-Flow allows to define timers which can be applied to either the template as a whole or to individual activities. There are four types of timers:

- *Absolute:* These timers are set to the absolute time when the timer will expire.
- *Calendar:* These timers start when the activity or process to which they are assigned becomes active. The time for this type of timer is counted using all seven days of the week and 24 hours of the day.
- *Business:* Similar to the previous ones they start when the activity or process to which they are assigned becomes active. The time for this type of timer is counted using only business days and hours.
- *Advanced:* These timers run according to a defined expression. The time for this type of timer can be counted using absolute time or only business days and hours depending on the expression entered.

Groupware aspect: In i-Flow the process or work-item owner can reassign activities. If every assignee declines, the activity becomes active for all of the other original assignees again.

Activities can be defined as voting activities, which use voting rules to determine the choice that wins. Voting Activities are designed so that many assignees can make their own choice on a particular Work Item. All of their choices (or votes) are polled, and a winning choice is made when any one of the voting rules is satisfied. Voting rules are assigned to a specific choice whereas the following types of rules can be assigned:

- *Majority Rule:* If this rule is assigned to a choice, a majority of votes for that choice make it the winning choice.
- *Percentage Rule:* If this rule is assigned to a choice, the specified percentage of votes for that choice would make it the winning choice.

- *"Number of" Rule*: If this rule is assigned to a choice, the specified number of votes for that choice would make it the winning choice.

4.2 Websphere MQ Workflow

Websphere MQ Workflow is a workflow management system that is part of the Websphere MQ product line of IBM. It is the successor of MQSeries Workflow, which itself is the successor of the IBM FlowMark system. Our evaluation is based on version 3.4 of the product, introduced in 2002.

Websphere MQ Workflow provides two separate tools: a build time environment for managing workflow models (templates), and a runtime environment for managing process instances. Both operate on separate databases; after a workflow has been modelled in the build-time, it has to be exported from the build-time database, and imported into the runtime database. The runtime environment can be accessed using a local Java client that connects to the MQ Workflow server using the Websphere MQ message queueing middleware, or using a browser that connects to a web-interface.

4.3 Functional aspect

Process models consists of activities, connected by control- and data connectors. Process models can be ordered per category. Process models can contain sub-processes; and every process model must contain a start and end node (called source and sink nodes). All activities are external, i.e. no standard activities are predefined but instead one has to use external activities for each user interaction.

4.4 Behavioural aspect

Process models in MQ Workflow are constructed graphically; one can connect activities using two kinds of arrows, control connectors and data connectors. Only one type of control connector is available for use, no nodes are used as in other workflow management systems.

The connectors pass tokens that contain either "true" or "false"; if an activity is passed a false token it should not execute but just pass the false token further; if an activity receives a true token it should execute and pass the token dependent on its execution. If an activity has multiple incoming arcs one can specify the boolean operation over those arcs, i.e. whether the activity should only execute if all tokens are true, or if it should execute if some tokens are true. By specifying this the activity behaves as an AND-join or as an OR-join.

One can also specify transition conditions on each control connector; the connector will only pass the "true" token if the condition is satisfied at runtime; the condition can be any boolean expression over the available workflow data at that connector. These transition conditions can be used to specify under which conditions an activity should execute, which is commonly implemented using a conditional node.

4.5 Informational aspect

Websphere MQ Workflow follows the principle of distinct data channels [13]. All data elements have to be mapped between activities explicitly. Data channels are specified with their own data connectors and can differ from the passing of control defined with the control connectors.

Data elements are modelled in so called data structures. These data structure definitions describe the contents of the input and output data containers of processes, activities, and blocks.

Any data that is used as input or output, or referred to in exit or transition conditions, must be described in a data structure definition. Each data structure consists of members. The data type of a data structure member can be either one of the basic MQ Workflow data types (VARIABLE LENGTH STRING, VARIABLE LENGTH BINARY, LONG, FLOAT) or can refer to another, previously defined data structure. A data structure that refers to another data structure is called a nested data structure, whereas the referred data structure can not link back to the source data structure to avoid cycles.

4.6 Operational aspect

In MQ Workflow activities can only consists of a subprocess (i.e. a composition of other activities) or execute an external application. The execution of external applications can be defined and reused across many activities in the workflow.

MQ Workflow invokes external applications through the operating system. Since the system is available on many different operating systems (e.g. Windows, Linux, UNIX, AIX, OS/360, etc.) the method of invocation and parameter passing varies per operating system. In general one indicates the external application using an operating system command (executable or dynamic loadable library). Parameters can be passed to external applications using workflow data that is visible at that activity in the workflow.

One can specify whether the external application is executed in the control sphere of the workflow management system, whether the application requires user intervention, whether the application is started automatically or has to be initiated by the user, and whether the workflow management system should wait for the termination of the application or if it should continue the process instance.

4.7 Organisational aspect

Websphere MQ Workflow supports modelling persons and their responsibilities in the organisation. One can model the properties of individuals, one can create roles and assign these roles to people, one can define organisations and attach people to them, and one can model security levels. It is further possible to assign levels to persons to distinguish them from each other. Usually one would assign the highest level to people with the most experience or with the greatest

skill. These levels can be used as a criterion to filter candidates for dynamic assignment to activities.

Every process and activity in the workflow model must be associated with one or more persons, identified by their user IDs. That is, every person referred to in a workflow model must already be defined in the Buildtime database. However, it is also possible to assign activities to roles, which means that there is no single person explicitly defined, but the activity is assigned to all role members. Any one of these persons can then perform the activity and when one user starts working on the task, the task is disabled for the other users.

When an instance of a process is started and the activity is ready to start, Websphere MQ Workflow uses the criteria for dynamic assignment to identify the group of possible starters for the activity.

A number of built-in roles are defined, such as “process starter”, that is assigned to the person that started the specific process instance; one could for instance specify that the process can only be finished by the same person that started it.

During runtime a suitable person is searched that can be assigned to an activity. Each activity can require some specific people, some specific role, some security level, or some organisation. It is also possible to model absences of people, and to require that only non-absent people are assigned to an activity.

One can also model how roles and authorisation can be inherited, or whether users are allowed to pass assignment of activities to other users.

5 Quantitative Analysis

The quantitative analysis describes the level of support in each workflow management systems for the control flow and data patterns from [2, 13]. We will first summarise these patterns; then we evaluate for each pattern whether it is supported in each workflow management system.

The control patterns are divided in several groups: basic patterns, advanced branching patterns, structural patterns, multiple instance patterns, state-based patterns, and cancellation patterns.

The basic patterns capture elementary control flow: a *sequence* of activities in which two activities execute after each other (meaning that the second one becomes enabled when the first one completes); a *parallel split (and-split)* and *parallel synchronisation (and-join)* of activities in which several activities execute in parallel (or in any order) and are later synchronised (the synchronise activity waits until all parallel branches are completed, and then continues); and an *exclusive choice (xor-split)* and *exclusive synchronisation (xor-join)* in which one out of several branches is chosen based on some condition and later synchronised (the synchronise activity waits until the active branch completes, since only one branch is assumed to be active).¹

¹although the synchronisation patterns are described independently from their split counterparts, it is assumed that they are used together, i.e. that a parallel split is followed by a parallel synchronisation.

The advanced branching patterns capture more complicated branching scenarios and are often not directly supported in existing systems: the *multi-choice (or-split)* in which one or more out of several branches is chosen based on some condition; such a split can be joined in three ways: the *synchronising merge* waits for all active branches of the split, and proceeds only after they have completed; the *discriminator* is similar, but the first active branch that completes already triggers the activity following the discriminator, all other active branches that complete later are ignored; the *multi-merge* joins the active branches without synchronisation, the activity following the multi-merge is started once for each active branch.

The structural patterns capture modelling situations that are usually forbidden in workflow management systems: *arbitrary cycles* are unstructured loops without predefined entry- and exit-points; *implicit termination* captures that a process should terminate implicitly when there are no activities left to perform.

The patterns involving multiple instances capture situations on which one case in a workflow has several child cases that are being instantiated in parallel. Each of these child cases needs to complete before the parent case can continue. The problems relate to being able to instantiate child cases from a parent case, and to being able to synchronise these instances and continue with the parent case after all child cases complete. For synchronisation the number of child cases is relevant, this number can be determined at design time or at runtime. If determined at run time it can be fixed before the instantiations start, or it can dynamically change during execution of the child cases. These parameters lead to different patterns: *multiple instances without synchronisation*, *multiple instances with design time knowledge*, *multiple instances with a-priori runtime knowledge*, and *multiple instances without a priori runtime knowledge*.

State-based patterns involve scenarios where an explicit notion of the state of the workflow process is relevant: the *deferred choice* is a split in which one out of several branches is chosen not by the workflow management system but by the environment; all branches are offered to the environment and the first one to be chosen invalidates the others; the *interleaved parallel routing* is an advanced sequential pattern, in which the order of the activities is arbitrary but the activities are not executed in parallel; the *milestone* pattern captures that an activity becomes enabled after completion of some activity but that it becomes disabled again if some other activity starts.

Cancellation patterns involve retracting running cases or executing activities: the *cancel activity* captures that some running activity can be cancelled by another activity or some external event; the *cancel case* captures that the whole running case can be cancelled by the execution of some activity or some external event.

The data flow are divided into four groups: data visibility patterns, data interaction patterns, data transfer patterns, and data-based routing patterns.

Data visibility patterns define the scope in which data elements are visible and capable of being utilised, which is primarily determined by the construct to which the data element is bound. Eight patterns are defined that relate to data

visibility. They range from the simple pattern where data elements are bound to a task (pattern 1), to data accessible in multiple instances of a single task (pattern 5), to environment data defined in the workflow system that can be accessed within a workflow instance (pattern 8).

Data interaction patterns examine the way data elements can be passed between components within a workflow process and are classified in two categories: internal and external data interaction. Six patterns describe the internal data interaction covering from the simple scenario where data flows between task instances (pattern 9), to more complex scenarios where data flows to or from a multiple instance task to a subsequent instance task (patterns 12-13). External data interaction patterns identify how external data is transferred between a component of the workflow and an external application. There are twelve external data interaction patterns, characterised by their dimensions: The type of workflow component that is interacting with the environment (task, case, workflow), whether the interaction is push or pull and what component constitutes the initiation role (the workflow component or the environment).

Data transfer patterns focus on the transfer of data elements between workflow components and on the mechanisms by which data elements can be passed between workflow components. There are seven distinct patterns in this category, ranging from transfer of the the value only (pattern 26 and 27) to patterns dealing with the possible interaction when applying a common data store (pattern 28, 29, 30) to patterns describing scenarios where data transformation functions are applied to a data element (pattern 31, 32).

The final set of patterns, called data-based routing patterns, capture the various ways in which data elements can interact with other workflow aspects. There are seven patterns in this category, ranging from scenarios where data-based preconditions (pattern 33 and 34) or postconditions (pattern 35 and 36) have to be checked prior to execution of a task to data-based routing (pattern 39) describing the ability to alter the control flow within a workflow case to trigger one or several subsequent task instances depending on an expression evaluating the values of data elements.

5.1 Control flow patterns

Pattern 1 (Sequence): Two or more activities in a workflow type are processed in the order, in which they were defined [2]. This most basic construct is achieved in i-Flow by connecting two activities with an arrow (see figure 1). In MQ Workflow this construct is achieved by connecting two activities with a control connector, see figure 2.

Pattern 2 (Parallel split): This pattern defines a point in a process which is split into several threads of control, all executed in parallel, whereas the order in which they are processed is not defined [2]. In i-Flow this pattern can be achieved by using the AND Node as shown in figure 3. In MQ Workflow this construct is achieved by having multiple outgoing control connectors from one activity: if the control connectors do not contain



Figure 1: Workflow Pattern 1 (Sequence) in i-Flow

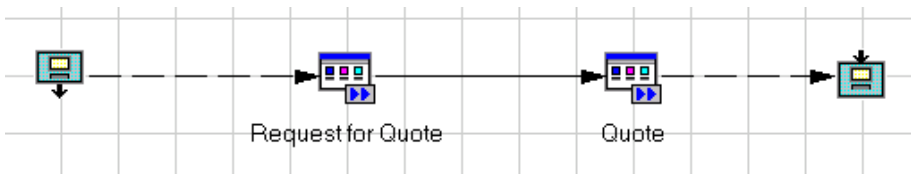


Figure 2: Workflow Pattern 1 (Sequence) in MQ Workflow

transition conditions they are by default all enabled, which means that all branches will be activated (see figure 4). In MQ Workflow no explicit control flow nodes exist, instead control flow constructs are implemented using transition conditions on the control connectors (arrows).

Pattern 3 (Synchronisation): A point where multiple parallel subprocesses converge into one thread, whereas each incoming branch is executed only once [2]. In i-Flow the AND node is used to merge any number of incoming branches. The process will only continue if every preceding activity is finished (see figure 5). In MQ Workflow no explicit synchronisation node is available or necessary, this construct is achieved by having multiple incoming control connectors to an activity. Multiple incoming control connectors mean that the activity is only enabled if all incoming control connectors are enabled, which means that all preceding activities have terminated (see figure 6).

Pattern 4 (Exclusive choice): It defines a point in the process where a certain condition based on a decision in the flow is taken [2]. i-Flow does not support an explicit XOR construct, but a conditional node to define exclusive transition conditions (see figure 7). In MQ Workflow a similar approach must be taken: a transition condition on the control connector defines which branches are activated; setting excluding transition conditions on two branches ensures an exclusive-choice behaviour. Transition conditions are not graphically visible, see figure 8.

Pattern 5 (Simple Merge): This pattern defines a point in the workflow process,

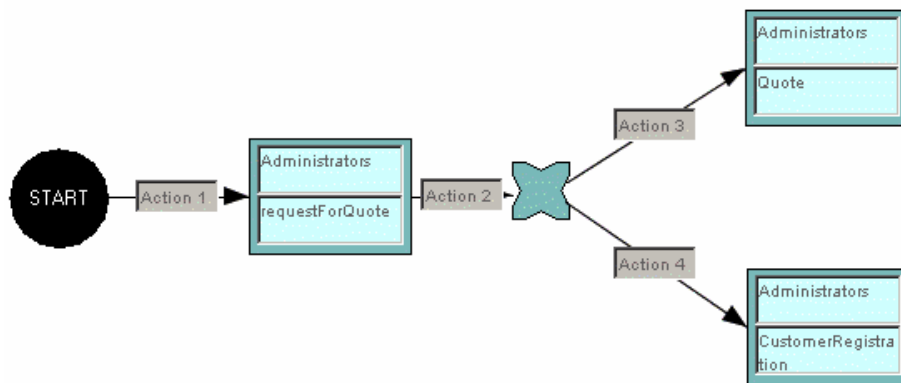


Figure 3: Workflow Pattern 2 (Parallel Split) in i-Flow

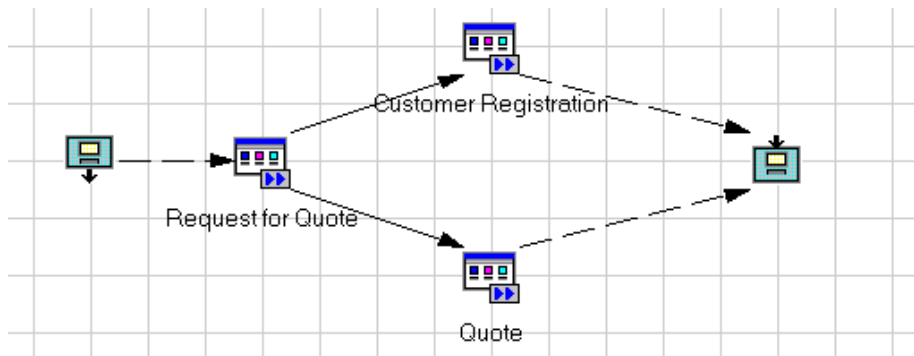


Figure 4: Workflow Pattern 2 (Parallel Split) in MQ Workflow

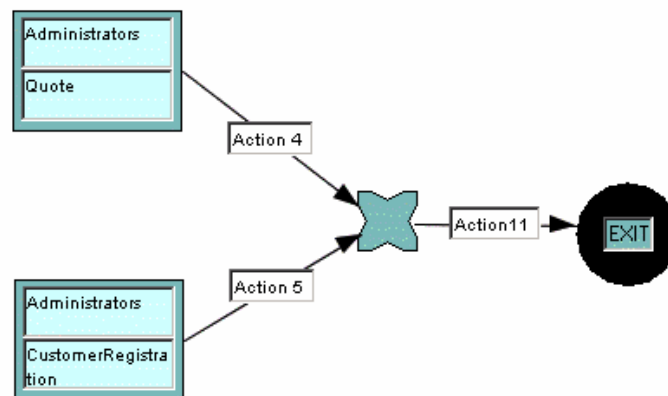


Figure 5: Workflow Pattern 3 (Synchronisation) in i-Flow

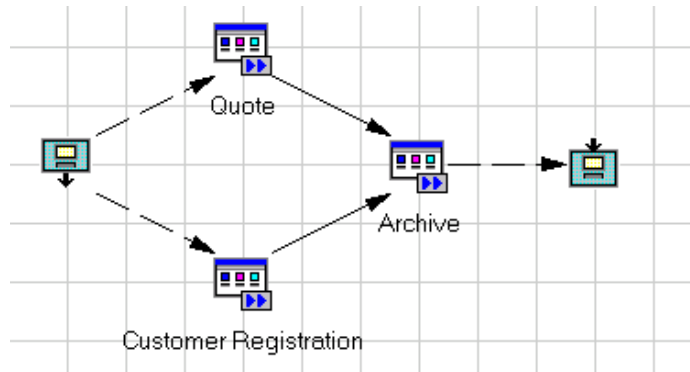


Figure 6: Workflow Pattern 3 (Synchronisation) in MQ Workflow

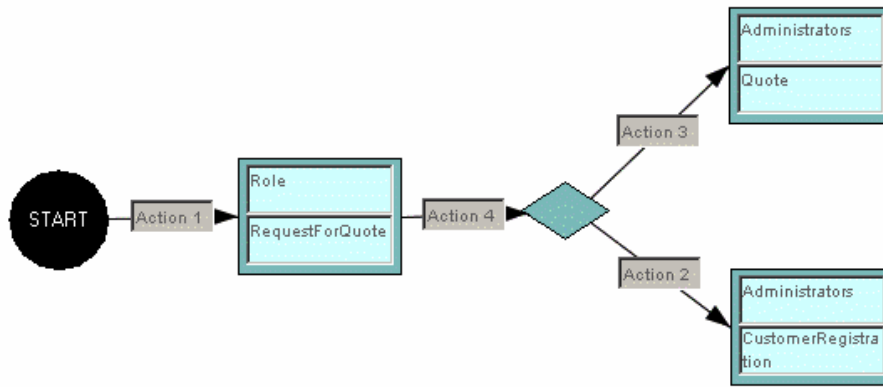


Figure 7: Workflow Pattern 4 (Exclusive choice) in i-Flow

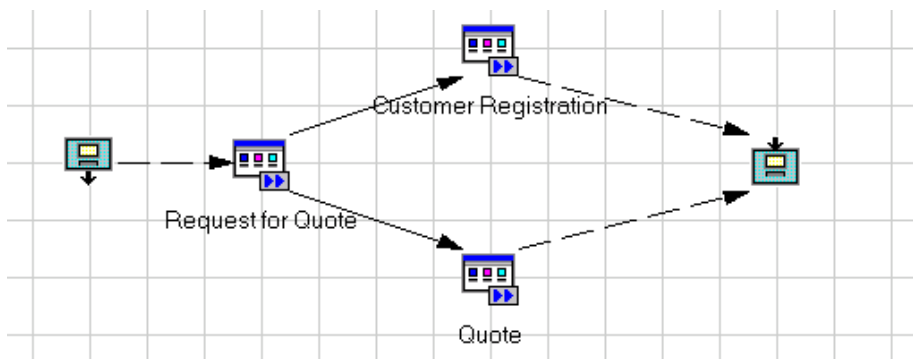


Figure 8: Workflow Pattern 4 (Exclusive choice) in MQ Workflow

where two or more alternative branches merge. The simple merge pattern does not support any kind of synchronisation, but it is assumed that only one of the branches is active) [2]. In i-Flow this pattern is achieved by using the OR node (see figure 9). In MQ Workflow this is achieved similar to the Pattern 3, with multiple incoming control connectors. In MQ Workflow synchronisation uses a technique of false-token propagation to ensure that synchronisation does not wait on inactive branches. Therefore synchronisation can also be used when not all branches are active, as in this pattern. Synchronisation is done implicitly by having more than one incoming control connectors to some activity.

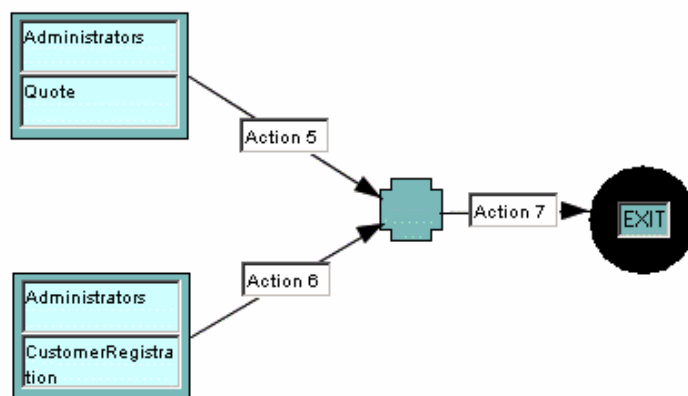


Figure 9: Workflow Pattern 5 (Simple Merge) in i-Flow

Pattern 6 (Multi-Choice): This pattern defines a point in the workflow process, where based on a decision a number of branches can be chosen [2]. For this pattern in i-Flow the same node can be used as for pattern 4, the conditional node (see figure 10). By defining transition conditions the paths to be taken are evaluated by the system, whereas the order of the evaluation can be defined as well. In MQ Workflow this pattern is also achieved similarly to Pattern 4, more than one outgoing control connector can be specified with transition conditions that determine whether a branch will be activated; see figure 11.

Pattern 7 (Synchronising Merge): This pattern defines a point in the process execution, where several branches merge into a single one. If more than one branch is active, the merge synchronises these branches (waits for all the active ones) [2]. This pattern is not directly supported by i-Flow. However since no simple OR is possible, there is no need for a synchronised merge. After a choice node always only one branch or all are active. The pattern is directly supported by MQ Workflow; as explained above, because of the special false-token propagation technique of MQ Workflow, all merges

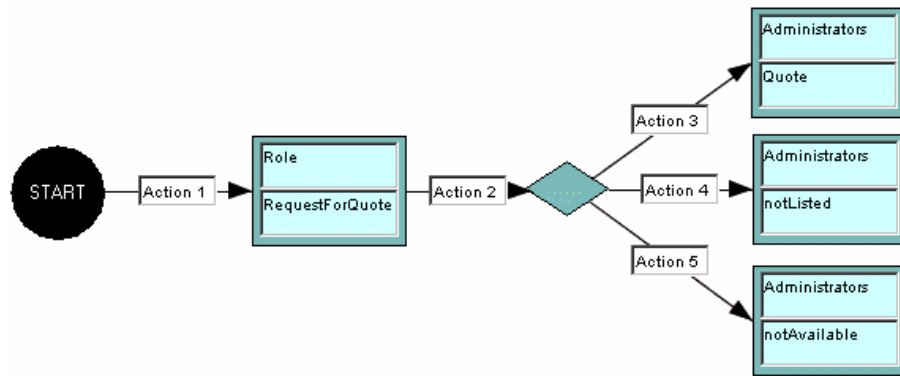


Figure 10: Workflow Pattern 6 (Multi-Choice) in i-Flow

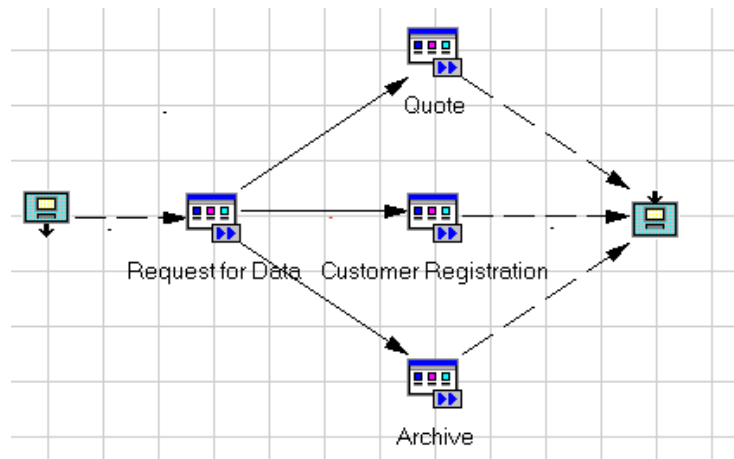


Figure 11: Workflow Pattern 6 (Multi-Choice) in MQ Workflow

are synchronising; they wait for all active branches but do not wait for inactive branches. See for example figure 6.

Pattern 8 (Multi-Merge): A point where two or more branches converge without synchronisation, but with multiple execution of the following activity if more than one branch is active [2]. This pattern is not supported by i-Flow. Activities following a merge will only execute once, never twice. The pattern is also not supported by MQ Workflow. All merges are synchronising, and thus activities following it will execute at most once.

Pattern 9 (Discriminator): This pattern describes a point in the workflow process which waits for completion of an incoming branch, before executing a subsequent flow of control. As the subsequent process is activated, the node remains active and ignores all other incoming branches [2]. i-Flow will never generate a second instance of the same activity if the first instance is still active. If a loop is defined after a choice, i-Flow will not generate a new activity if it is still active from before the iteration. In MQ Workflow this pattern is also not supported, since any merge is always synchronising, and thus waits for all incoming control connectors before continuing.

Pattern 10 (Arbitrary Cycles): This pattern defines a point in the business process, where a portion of the process can be repeated [2]. In i-Flow arbitrary cycles can easily be achieved by use of an AND node with an outgoing branch and two incoming (see figure 12). Again, if the AND node used for the merging has more than one outgoing branch, activities that are still active after one cycle has completed are not initiated again. In MQ Workflow only structured cycles (that have exactly one entry and exit point) are allowed; these can be achieved using the Data Loop connector. Arbitrary cycles are not possible in MQ Workflow, jumping back to some point in the workflow is explicitly forbidden.

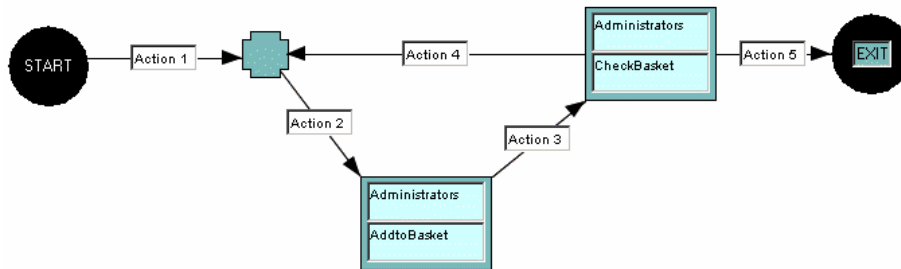


Figure 12: Workflow Pattern 10 (Arbitrary Cycles) in i-Flow

Pattern 11 (Implicit Termination): An executed sub process is automatically terminated, when there is nothing left to be done [2]. i-Flow does not

support this pattern, since all activities are terminated when a final node is reached, regardless if the workflow type has a number of final nodes with branches possibly still active. In MQ Workflow this pattern is supported: if no activities in the workflow can be reached anymore the process instance is terminated, even if the explicit exit node is not reached yet.

Pattern 12 (Multiple Instances without synchronisation): Within a single case multiple instances of an activity can be created, whereas all of them are independent of each other and no synchronisation is needed [2]. i-Flow supports this pattern with the Chained Process Node. By using this construct a sub-process is split off the main branch and executed concurrently (see figure 13). MQ Workflow does not support this pattern, since activities can not “spawn-off” from the main branch and execute independently.



Figure 13: Workflow Pattern 12 (Multiple instances without synchronisation) in i-Flow

Pattern 13 (Multiple Instances with a priori design time knowledge): An activity within a process instance is enabled multiple times, whereas the number of execution is known at design time [2]. This simple pattern is achieved similarly to Pattern 2 by using the AND node and defining the number of branches at design time (see figure 14). This pattern is not supported by MQ Workflow, although a workaround is possible by creating a parallel split (as described in Pattern 2) that contains in each branch the same activity, see figure 15

Pattern 14 (Multiple Instances with a priori runtime knowledge): In this pattern an activity is enabled multiple times in one workflow case. It is known at runtime how many instances of the activity has to be enabled before the process continues [2]. As i-Flow does not allow enable activities multiple times this pattern is not supported. This pattern is not supported in MQ Workflow; a workaround is possible using the exit conditions of an activity: the activity is repeated until the exit condition is satisfied. However, the multiple instances patterns describe activities that are performed in parallel, which is not the case in this workaround.

Pattern 15 (Multiple Instances without a priori runtime knowledge): Similar to Pattern 14 an activity is enabled multiple times, but in this case with the difference that only at the each initiation of the activity it can be determined if another instances has to be created [2]. Again, i-Flow does not

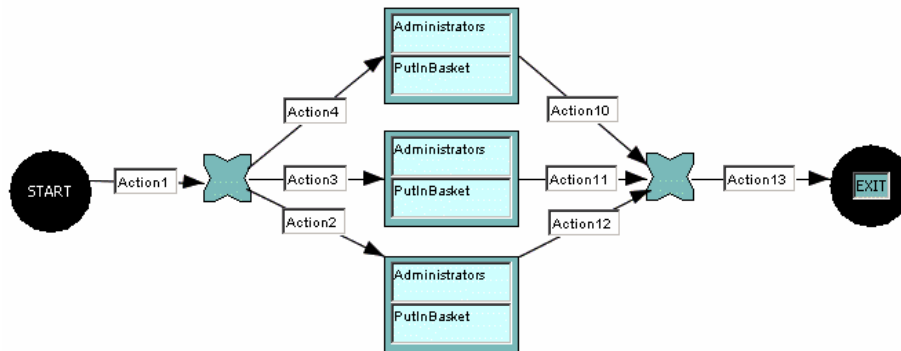


Figure 14: Workflow Pattern 13 (Multiple instances with a priori design time knowledge) in i-Flow

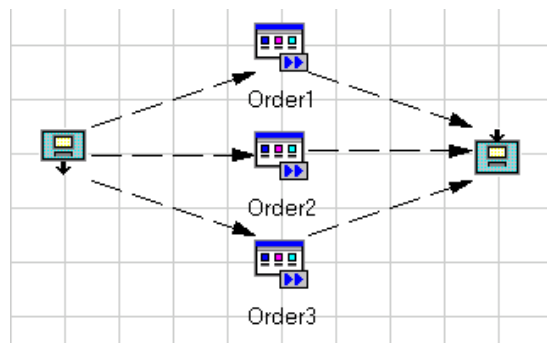


Figure 15: Workflow Pattern 13 (Multiple instances with a priori design time knowledge) workaround in MQ Workflow

support multiple instantiations of a single activity and therefore does not support this pattern. Similar to Pattern 14, this pattern is not supported by MQ Workflow; the same workaround could be used but the activities are then not performed in parallel.

Pattern 16 (Deferred Choice): Similar to the multi choice pattern, one of several branches is chosen. In contrast the choice for one branch is not made by the workflow management system, but all branches are offered to the participants; when a participant chooses one branch all other branches are cancelled [2]. The choice is deferred until the latest possible moment. In i-Flow this pattern is not directly supported. However it is possible to use the AND node and cancel all other branches with a Java action. This pattern is not supported by MQ Workflow; a split is either parallel, in which case all branches are offered and need to be performed, or exclusive, in which case the choice for a specific branch is made by the workflow management system.

Pattern 17 (Interleaved parallel routing): This pattern defines a point in execution, where a set of activities is processed in an arbitrary order. Each activity is executed only once, but the order in which they are processed is defined at run-time. Neither i-Flow nor MQ Workflow support this pattern. However as shown in [2] a cluttered workaround is possible by using a conditional node with branches covering all possible sequences.

Pattern 18 (Milestone): The Milestone defines a point in the workflow, where a determined milestone has to be reached to enable a given activity. The milestone can also expire which means that the activity will not be enabled [2]. In i-Flow this pattern can only be achieved in a programmatic way, by disabling activities in the database. In MQ Workflow this pattern is not supported; although it is possible to enable an activity when the milestone is reached, there is no way to disable the activity when the milestone has passed.

Pattern 19 (Cancel Activity): An activity is disabled after it has been already enabled [2]. There is no direct support in i-Flow for this pattern, but again a programmatic workaround could be implemented. The pattern is not supported in MQ Workflow.

Pattern 20 (Cancel Case): A complete workflow instance is removed [2]. Again only a programmatic solution can achieve the cancelling of a workflow case in i-Flow. Again the pattern is not supported in MQ Workflow.

5.2 Data Patterns

Pattern 1 (Task Data): Data element can be defined for a task exclusively accessible within the individual execution of the task. Data in i-Flow is visible in all activates within the workflow case. Hence all data elements

are available to be used in any task. MQ Workflow supports to restrict the scope of a data structure to one task.

Pattern 2 (Block Data): Block Tasks are able to define data elements which are accessible by each activity in the corresponding sub-workflow [13]. In i-Flow data elements have to be mapped to and from the sub-workflow to its respective super-workflow. The same applies for data elements defined in the sub-workflow, which have to be mapped to the super-workflow as well to be used there. In MQ Workflow data elements have to explicitly be passed to sub-workflows. However a global container node in the sub-workflow can be used to store data elements that are shared between all activities in the sub-process.

Pattern 3 (Scope Data): Data elements which are accessible by a subset of the task in a case are called scope data [13]. It is not possible to define data at a scope level in i-Flow, since all data elements as mentioned are accessible by all activities in a workflow case. In Workflow MQ it is not possible to restrict the visibility of data elements to block tasks.

Pattern 4 (Multiple Instance Data): Multiple Instance Data denotes data elements which are specific to only one of multiple executions of an activity. Since i-Flow does not support multiple instances at the control flow level, multiple instance data is not supported either. In MQ Workflow one can use a data loop connector which specifies the flow of data from the output container back to the input container of the same activity. In this way multiple instances can be executed sequentially with data elements specific to the respective execution.

Pattern 5 (Case Data): Case Data are data elements which are specific to workflow case or to one of its instances [13]. As mentioned all data in i-Flow is case data and available to all its respective activities. i-Flow implements a data management with a common data store for each workflow case, although individual data fields must be passed to sub-workflows in order to make them accessible to the tasks within them (see pattern 2). In MQ Workflow the same node as for pattern 2, the global container node, can be applied to make data elements available to all components during execution.

Pattern 6 (Workflow Data): Since i-flow applies a common data store this pattern underlies the conceptual model of the system. However as mentioned above for sub-workflows each data element has to be mapped to it. Workflow data in its interpretation of [13] is therefore not supported.

Pattern 7 (Environment Data): Environment data denotes data elements which are external to the workflow environment. i-flow supports different ways to map external data to its internal User defined attributes (UDA). One way is to use built-in Java script commands (e.g. Set UDA value from SQL) for exchanging information with the Server. Generic Java Actions

can be applied to request data from any external data source. In Web-sphere MQ access to external sources have to be achieved by the program called within every individual program activity.

Pattern 8 (Data Interaction - Task to Task): This pattern defines the way data elements are passed between two task instances [13]. As mentioned data in i-Flow is shared via a global data store, hence no explicit data passing is required except between super- and sub-workflow. Workflow MQ employs distinct data channels where data structures are passed between the respective nodes.

Pattern 9 (Data Interaction - Block Task to Sub-Workflow Decomposition): It defines the ability to pass data elements from a block task instance to the corresponding sub-workflow [13]. In i-Flow data elements supplied in the block task must be specifically passed as parameters to the underlying sub-workflow implementation. In Workflow MQ data structures are passed to the sub-workflow by use of the source and sink node.

Pattern 10 (Data Interaction - Sub-Workflow Decomposition to Block Task): The ability to pass block task data back from the sub-workflow to the super-workflow [13]. Similar to the other way described in pattern 9 i-Flow requires a strict mapping from variables defined in the sub-workflow to the block task event in order to avoid that data mismatches occur. Similar to pattern 9 in Workflow MQ data structures are passed from the sub-workflow by use of the source and sink node as well.

Pattern 11 (Data Interaction - to Multiple Instance Task): This pattern defines the ability to pass data from one task to another task which is supporting multiple instances [13]. Multiple instances are not directly supported within i-Flow and Workflow MQ, hence this pattern is not supported.

Pattern 12 (Data Interaction - from Multiple Instance Task): Similar to Pattern 11, but the opposite way, this pattern defines the passing from data from a task supporting multiple instances to a subsequent task [13]. For the i-Flow and Workflow MQ support of this task, the same as for Pattern 11 applies.

Pattern 13 (Data Interaction - Case to Case): The passing of data between two workflow cases at runtime [13]. In i-Flow there is no direct way to pass data between two workflow cases at runtime. It can be indirectly achieved via an external database. In Workflow MQ the same workaround is possible, but no direct support is provided.

Pattern 14-17 (Data interaction - Task/Environment): [13] defines four different patterns to describe the way how one task can pass data elements to an external source and the other way round. The break down into Pull-oriented and Push-Oriented is in our opinion artificial and we subsume it to one. As described in pattern 7 i-Flow supports Java Actions to retrieve

data from external sources, where it is possible to optionally pass an enactment context object, which constitutes an API that the Java code can use as a handle to get information out of process instances. The method can get and set any process variable (e.g. process name, ID, description, priority etc.) and User-defined attribute values. MQ Workflow provides application program interface (API) and Extensible Markup Language (XML) message interface support. All persistent objects in Workflow MQ, among them also data structures, can be accessed from the server. To set data structures a container value has to be set. The container can then be used as the input container for a process instance or as the output container of a work item.

Pattern 18-21 (Data Interaction - Case/Environment): This patterns denote the ability to pass data element between cases and external sources [13]. Again we do not make a distinction between Push- and Pull-oriented. In i-Flow and Websphere MQ there is no possibility to pass data on case level. However in i-Flow variables are stored in a common data-store and accessible by all tasks in a workflow.

Pattern 22-25 (Data Interaction - Workflow/Environment): The ability of a workflow engine to pass data elements to external sources [13]. The difference to the four previous patterns is that the data passing is independent of any tasks or workflow case. These patterns are not supported in i-Flow.

Pattern 26-27 (Data Transfer by Value - Incoming/Outgoing): This pattern describes the ability of workflow components to receive (Incoming Data Transfer) or pass data elements (Outgoing Data Transfer) by value [13]. Both patterns are not necessary in i-Flow, since it is using the common data store. In Websphere MQ data values are passed when mappings between data structures are defined.

Pattern 28 (Data Transfer - Copy In/Copy Out): This denotes the ability to copy data values of a set of data elements to an common address space at commencement and copy their final values back at completion [13]. i-Flow supports no local variables in tasks. However a data element with different name can be mapped with the value of another data element in the prologue and copied back in the epilogue of a task. Websphere MQ does not support local copies.

Pattern 29 (Data Transfer by Reference - Unlocked): It describes the ability to address data elements by passing reference to a location known to both components [13]. Again this is not necessary in i-Flow as it uses a common data store with named data location. In Websphere MQ this pattern is not supported.

Pattern 30 (Data Transfer by Reference - With Lock): This denotes the same as Pattern 29 with the exception that the data is locked only to the receiving

component [13]. Data elements are not locked in i-Flow in the common data store. For Websphere MQ the same as for pattern 29 applies.

Pattern 31-32 (Data Transformation - Input/Output): This pattern defines the ability to apply transformation functions to a data element prior or after its passing to a workflow component [13]. In i-Flow triggers allow to map external data from XML files to UDAs. By loading a XML schema file, mappings can be defined from each element to defined UDAs for the activity. Websphere MQ Workflow does not allow data transformations prior to its passing to a workflow component.

Pattern 33 (Task Precondition - Data Existence): Data-based preconditions can be specified for tasks based on the presence of data elements at runtime [13]. i-Flow provides the ability to set default values for fields which do not have a value recorded for them. Conditional nodes can be used to route the control flow around a task. Workflow MQ also allows to set default values for data elements at design time.

Pattern 34 (Task Precondition - Data Value): Similar to the former pattern, but with a precondition on the value of a data element [13]. i-Flow provides a conditional node which allows to route the control flow around tasks where data preconditions are not met. There is no pre-condition checking within the task itself. There is no support for this pattern in Workflow MQ.

Pattern 35 (Task Postcondition - Data Existence): It describes if data-based postconditions can be specified at task level to check the existence of specific parameters at runtime [13]. This pattern is not supported in i-flow. In Workflow MQ one can define an exit condition for an activity that causes the activity to be repeated until the condition is met. Depending on the evaluation of the exit condition, the activity either reaches finished status or returns to ready status.

Pattern 36 (Task Postcondition - Data Value): Similar to Pattern 35 data-based postconditions can be specified, but here based on the value of specific parameters at execution time [13]. This pattern is not directly supported by the data aspect in i-Flow itself either. The task has to pass the control to a subsequent Conditional Node to assess the value and can direct the control flow back to the start of the prior activity. In Workflow MQ exit conditions can evaluate the value of data elements as described in pattern 35 for data existence.

Pattern 37 (Event-based Task Trigger): It defines the ability for an external component to initiate a task in the workflow model [13]. i-Flow allows to initiate tasks by an external API. Through this it is possible to start or edit sessions or single tasks. Another way to pro-actively access the workflow type is to define triggers at design time to cause processes or tasks to be

started by external events. in Websphere MQ Workflow activity triggering is supported via various APIs.

Pattern 38 (Data-based Task Trigger): This pattern denotes the ability of a workflow system to trigger a specific task based on the evaluation of an expression prior to the task [13]. i-Flow allows to define Triggers which can evaluate if a specific data element value fulfills a conditional expression and subsequently start the task. Websphere MQ Workflow does not support this pattern.

Pattern 39 (Data-based Routing): It defines the ability to alter the control flow within a workflow model based on the value of data-based expressions [13]. i-Flow provides a conditional node in which conditional expressions can be defined and evaluated in runtime. Workflow MQ directly supports this pattern with transition conditions, which define a logical expression associated with a conditional control connector. If specified, it must be true for control to flow along the associated control connector.

6 Conclusion

There is nothing to conclude so far.

References

- [1] W. M. P. van der Aalst. The application of Petri nets to workflow management. *The Journal of Circuits, Systems and Computers*, 8(1):21–66, 1998.
- [2] W. M. P. van der Aalst, A. H. M. ter Hofstede, B. Kiepuszewski, and A. P. Barros. Workflow patterns. *Distributed and Parallel Databases*, 14(1):5–51, 2003.
- [3] N. E. Fenton and S. L. Pfleeger. *Software Metrics: A Rigorous and Practical Approach*. Thomson Computer Press, 1996.
- [4] D. Georgakopoulos, M. Hornick, and A. Sheth. An overview of workflow management: from process modeling to workflow automation infrastructure. *Distributed and Parallel Databases*, 3(2):119–153, 1995.
- [5] L. J. Hommes. *The Evaluation of Business Process Modeling Techniques*. Ph.D. thesis, Delft University of Technology, 2004.
- [6] S. Jablonski and C. Bussler. *Workflow Management: Modeling Concepts, Architecture and Implementation*. International Thomson Computer Press, 1996.

- [7] B. A. Kitchenham. Evaluating Software Engineering Methods and Tool. Part 1: The Evaluation Context and the Evaluation Methods. *ACM SIGSOFT Software Engineering Notes*, 21:11–15, 1996.
- [8] R. Marschak. Perspectives on workflow. In T. White and L. Fischer, (eds.) *New Tools for New Times: The Workflow Paradigm*. Future Strategies Inc., California, USA, 1994.
- [9] M. zur Muehlen. Organizational management in workflow applications – issues and perspectives. *Information Technology and Management*, 5:271–291, 2004.
- [10] M. zur Muehlen. *Workflow-based Process Controlling*. Logos Verlag Berlin, 2004.
- [11] S. Mukherjee, *et al.* Logic-based approaches to workflow modeling and verification. In J. Chomicki, R. van der Meyden, and G. Saake, (eds.) *Logics for Emerging Applications of Databases*, chap. 5, pp. 167–202. Springer-Verlag, Berlin, 2004.
- [12] S. Petkov, E. Oren, and A. Haller. Aspects in workflow management. Tech. Rep. DERI-TR-2005-04-10, Digital Enterprise Research Institute (DERI), 2005.
- [13] N. Russell, A. H. M. ter Hofstede, D. Edmond, and W. M. P. van der Aalst. Workflow data patterns. Tech. Rep. FIT-TR-2004-01, Queensland University of Technology, Brisbane, 2004.
- [14] A. P. Sheth, W. M. P. van der Aalst, and I. B. Arpinar. Processes driving the networked economy. *IEEE Concurrency*, 7(3):18–31, 1999.