WORKFLOW DATA PATTERNS

Nick Russell[1], Arthur H.M. ter Hofstede[1], David Edmond[1]
[1] *Centre for Information Technology Innovation, Queensland University of Technology*
GPO Box 2434, Brisbane QLD 4001, Australia
{n.russell,a.terhofstede,d.edmond}@qut.edu.au

Wil M.P. van der Aalst[1,2]
[2] *Department of Technology Management, Eindhoven University of Technology*
GPO Box 513, NL-5600 MB Eindhoven, The Netherlands
w.m.p.v.d.aalst@tm.tue.nl

Abstract

Workflow systems seek to provide an implementation vehicle for complex, recurring business processes. Notwithstanding this common objective, there are a variety of distinct features offered by commercial workflow management systems. These differences result in significant variations in the ability of distinct tools to represent and implement the plethora of requirements that may arise in contemporary business processes. Many of these requirements recur quite frequently during the requirements analysis activity for workflow systems and abstractions of these requirements serve as a useful means of identifying the key components of workflow languages.

Previous work has identified a number of *workflow control patterns* which characterise the range of control flow constructs that might be encountered when modelling and analysing workflow. In this paper, we describe a series of *workflow data patterns* that aim to capture the various ways in which data is represented and utilised in workflows. By delineating these patterns in a form that is independent of specific workflow technologies and modelling languages, we are able to provide a comprehensive treatment of the workflow data perspective and we subsequently use these patterns as the basis for a detailed comparison of a number of commercially available workflow management systems and business process modelling languages.

Keywords: Patterns, Data, Workflow, Business Process Modelling

1 Introduction

There are a series of concepts that apply to the representation and utilisation of data within workflow systems. These concepts not only define the manner in which data in its various forms can be employed within a business process and the range of informational concepts that a workflow engine is able to capture but also characterise the interaction of data elements with other workflow and environmental constructs.

Detailed examination of a number of workflow tools and business process modelling paradigms suggests that the way in which data is structured and utilized within these tools has a number of common characteristics. Indeed these characteristics

bear striking similarities to the *Workflow Patterns* [AHKB03] and *Design Patterns* [GHJV95] initiatives in terms of their generic applicability across the class of workflow engines, except in this case, they refer specifically to the data perspective of workflow systems and the manner in which it interrelates with other workflow perspectives.

The use of a patterns-based approach for illustrating data-related concepts in workflow systems offers the potential to describe these concepts in a language-independent way. This ensures that the patterns identified have broad applicability across a wide variety of workflow implementations. It provides the basis for comparison of data-related capabilities between distinct products and offers a means of identifying potential areas of new functionality in workflow systems. In this paper we focus on workflow technology but the results identified apply to any process-aware information system (PAIS).

1.1 Background and related work

Interest in workflow systems has grown dramatically over the past decade and is evidenced by the uptake in membership of industry bodies such as the Workflow Management Coalition [Wor04] which now lists 270 vendors, associate and academic members. This has fuelled the development of a multitude of workflow engines each with unique features and capabilities, yet despite attempts by industry bodies such as the Workflow Management Coalition [Hol95, Wor02] and the Object Management Group [OMG00] to provide standards for workflow management systems, there is limited adoptance by commercial vendors.

Perhaps the most significant shortcoming in this area is the absence of a common formalism for workflow modelling. A number of possible candidates have been considered from the areas of process modelling and general systems design including Petri-Nets [Aal96, Aal98], Event-Driven Process Chains (EPCs) [Sch00] and UML Activity Diagrams [DH01] although none of these have achieved broad usage.

One of the major impediments to a generic modelling technique is the broad range of offerings that fall under the "workflow umbrella" [GHS95] – ranging from unstructured groupware support products through to transaction-oriented production workflows – and the inherent difficulty of establishing a conceptual framework that is both suitable and meaningful across the entire range of offerings.

The recent *Workflow Patterns* initiative [AHKB03] has taken an empirical approach to identifying the major control constructs that are inherent in workflow systems through a broad survey of process modelling languages and software offerings. The outcomes of this research were a set of twenty patterns characterising commonly utilised process control structures in workflow systems together with validation of their applicability through a detailed survey of thirteen commercial workflow products and two research prototypes.

One significant advantage of a patterns-based approach is that it provides a basis for comparison between software offerings without requiring that they share the same conceptual underpinnings. This paper aims to extend the previous work on *Workflow Control Patterns* to the data perspective [JB96]. It identifies 39 data patterns that recur in workflow systems and examines their use across six major workflow products and web service offerings.

1.2 Structure of this paper

This paper proceeds as follows. Section 2 provides a taxonomy of the common data characteristics or patterns which have been identified in the context of workflow systems. Section 3 relates these patterns to a number of commercial workflow systems and process modelling languages. Section 4 discusses the results of these investigations and potential future research directions. Section 5 outlines related material in the patterns, business process modelling and workflow systems areas and Section 6 concludes the paper. A number of appendices are included which explain the ratings achieved by each of the workflow products for specific data patterns and outline the overall patterns assessment criteria.

2 Data characterisation

From a data perspective, there are a series of characteristics that occur repeatedly in different workflow modelling paradigms. These can be divided into four distinct groups:

- **Data visibility** — relating to the extent and manner in which data elements can be viewed by various components of a workflow process.
- **Data interaction** — focussing on the manner in which data is communicated between active elements within a workflow.
- **Data transfer** — which consider the means by which the actual transfer of data elements occurs between workflow components and describe the various mechanisms by which data elements can be passed across the interface of a workflow component.
- **Data-based routing** — which characterise the manner in which data elements can influence the operation of other aspects of the workflow, particularly the control flow perspective.

Each of these characteristics are examined in more detail in the following sections.

For the purposes of this discussion, we take a broad view of the potential structure and substance of workflow data. The representational capabilities of workflow systems vary widely and the more sophisticated of them enable data elements of significant complexity to be captured by including provision for composition, repetition and reference operators within the data specification language thus providing support for the construction and use of abstract data types for managing workflow data.

Table 1 identifies the major structural characteristics that are relevant to the specification of data elements and indicates their applicability to individual workflow engines.

Construct	**Staffware**	**MQSeries**	**FLOWer**	**COSA**	**XPDL**	**BPEL4WS**
string	•	•	•	•	•	•
integer	•	•	•	•	•	•
float	•	•	•	•	•	•

Construct	Staffware	MQSeries	FLOWer	COSA	XPDL	BPEL4WS
boolean			•	•	•	•
date	•		•	•	•	•
time	•		•	•		•
document/reference	•		•	•	•	•
enumeration					•	•
composition	•	•	•		•	•
array		•	•		•	
set						•

Table 1: Data Elements in Workflow Systems

For the purposes of this paper, we do not examine the ability of workflow engines to describe arbitrary classes of data structures but focus rather on how these elements are utilised in the context of workflow operation.

2.1 Workflow structure

Before we describe the actual data patterns in detail, we first present a standard set of definitions for the various components of a workflow system that we will utilise throughout this paper.

A *workflow* or *workflow model* is a description of a business process in sufficient detail that it is able to be directly executed by a *workflow management system*. A *workflow model* is composed of a number of *tasks* which are connected in the form of a directed graph. An executing instance of a workflow model is called a *case* or *process instance*. There may be multiple cases of a particular workflow model running simultaneously, however each of these is assumed to have an independent existence and they typically execute without reference to each other.

There is usually a unique first task and a unique final task in a workflow. These are the tasks that are first to run and last to run in a given workflow case. Each invocation of a task that executes is termed a *task instance*. A task instance may initiate one or several task instances when it completes. This is illustrated by an arrow from the completing task to the task being initiated e.g. in Figure 1, task instance B is initiated when task instance A completes. This may also occur conditionally and where this is the case, the edge between task instances indicates the condition that must be satisfied for the subsequent task instance to be started e.g. task instance D is initiated when task instance C completes if the data element M is greater than 5.

A *task* corresponds to a single unit of work. Four distinct types of task are denoted: *atomic*, *block*, *multi-instance* and *multi-instance block*. We use the generic term *components* of a workflow to refer to all of the tasks that comprise a given workflow model.

An *atomic task* is one which has a simple, self-contained definition (i.e. one that is not described in terms of other workflow tasks) and only one instance of the task executes when it is initiated.

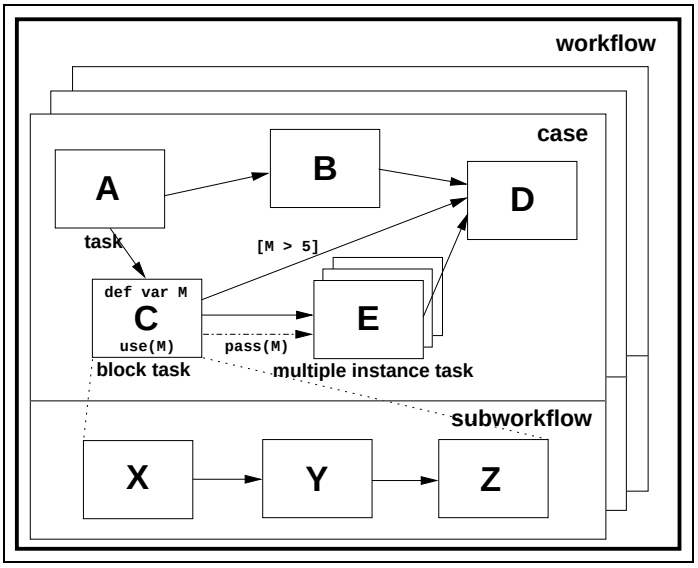

Figure 1: Components of a workflow

A *block task* is a complex action which has its implementation described in terms of a *sub-workflow*. When a *block task* is started, it passes control to the first task(s) in its corresponding *sub-workflow*. This *sub-workflow* executes to completion and at its conclusion, it passes control back to the *block task*. E.g. block task C is defined in terms of the sub-workflow comprising tasks, X, Y and Z.

A *multiple-instance task* is a task that may have multiple distinct execution instances running concurrently within the same workflow case. Each of these instances executes independently. Only when a nominated number of these instances have completed is the task following the multiple instance task initiated.

A *multi-instance block task* is a combination of the two previous constructs and denotes a task that may have multiple distinct execution instances each of which is block structured in nature (i.e. has a corresponding *sub-workflow*).

The control flow between tasks occurs via the *control channel* which is indicated by a solid arrow between tasks. There may also be a distinct *data channel* between workflow tasks which provides a means of communicating *data elements* between two connected tasks. Where a distinct data channel is intended between tasks, it is illustrated with a broken (dash-dot) line between them as illustrated in Figure 1 between task instances C and E. In other scenarios, the control and data channels are combined, however in both cases, where data elements are passed along a channel between tasks, this is illustrated by the `pass()` relation, e.g. in Figure 1 data element M is passed from task instance C to E.

The definition of data elements within the workflow is illustrated by the `def var` *`variable-name`* phrase. Depending on where this appears, the variable may have task, block, case or workflow scope indicating the level at which the data element is bound. The places where a given data element can be accessed are illustrated by the `use()` phrase. In the case of workflow, case and block level data elements, these may be passed between tasks by reference (i.e. the location rather than the value of the data element is passed). This is indicated through the use of the `&` symbol e.g. the

`pass(&M)` phrase indicates that the data element M is being passed by reference rather than value.

The patterns presented in this paper are intended to be language independent and do not assume a concrete syntax. In the absence of an agreed workflow model, the aim is to define them in a form that ensures they are applicable to the broadest possible range of workflow systems. As such, we use informal diagrams throughout this paper for illustrating workflow execution instances. The aim of these diagrams is to illustrate the scope of workflow data and the manner in which it is passed between various workflow components. They do not have a formal semantics for the control perspective. Where this is required, we use the YAWL notation [AH04] which provides a more rigorous means of describing workflow control structures.

2.2 Data visibility

Within the context of a workflow engine, there are a variety of distinct ways in which data elements can be defined and utilised. Typically these variations relate to the manner in which they are declared and the main workflow construct to which they are anchored. More importantly, they directly influence the way in which the data element may be used e.g. to capture *production* information, to manage *control* data or for *communication* with the external environment. In this section, we consider each of the potential contexts in which a data construct can be defined and utilised.

Pattern 1 (Task Data)

Description Data elements can be defined by tasks which are accessible only within the context of individual execution instances of that task.

Example

- The *working trajectory* variable is only used within the *Calculate Flight Path* task.

Motivation To provide data support for local operations at task level. Typically these data elements will be used to provide working storage during task execution for control data or intermediate results in the manipulation of production data.

Figure 2 illustrates the declaration of a task data element (variable X in task B) and the scope in which it can be utilised (shown by the shaded region and the `use()` function). Note that it has a distinct existence (and potential value) for each instance of task B (i.e. in this example it is instantiated once for each workflow case since task B only runs once within each workflow).

Implementation The implementation of task data in a workflow system takes one of two forms - either data elements are defined as parameters to the task making them available for use within the task or they are declared within the definition of the task itself. In either case, they are bound in scope to the task block and have a lifetime that corresponds to that of the execution of an instance of that task.

Issues One difficulty that can arise is the potential for a task to declare a data element with the same name as another data element declared elsewhere (either within another task or at a different level within the workflow hierarchy (e.g. block, case, process level) that can be accessed within the task.

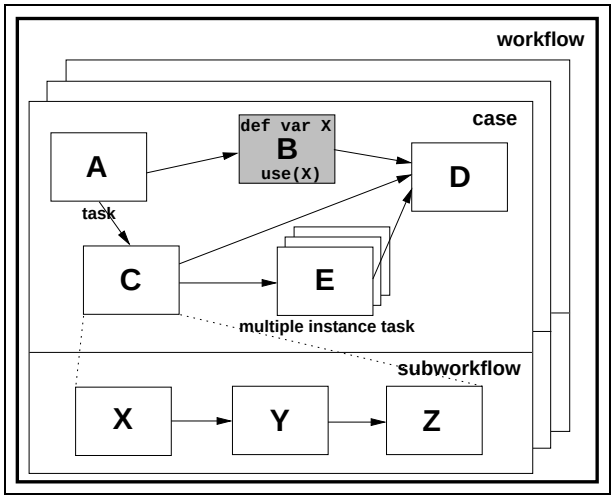

Figure 2: Task level data visibility

A second issue that may require consideration can arise where a task is able to execute more than once (e.g. in the case of a multi-merge [AHKB03]). When the second (or later) instance commences, should the data elements contain the values from the first instance or should they be re-initialised.

Solutions The first issue can be addressed through the use of a tight binding approach at task level restricting the use of data elements within the task to those explicitly declared by the task itself and those which are passed to the task as formal parameters. All of these data element names must be unique within the context of the task.

An alternative approach is employed in BPEL4WS, which only allows access to the data element with the innermost scope in the event of name clashes. Indeed, this approach is proposed as a mechanism of 'hiding' data elements from outer scopes by declaring another with the same name at task level.

The second issue is not a major consideration for most workflow tools which initialise data elements at the commencement of each task instance. One exception to this is FLOWer which provides the option for a task instance which comprises part of a loop in a workflow to either refresh data elements on each iteration or to retain their values from the previous instance (in the preceding loop iteration). COSA also caters for both situations through the use of distinct classes of variables (ACTIVITY and STD).

Pattern 2 (Block Data)

Description Block tasks (i.e. tasks which can be described in terms of a corresponding sub-workflow) are able to define data elements which are accessible by each of the components of the corresponding sub-workflow.

Example

- All components of the sub-workflow which define the *Assess Investment Risk* block task can utilise the *security details* data element.

Motivation The manner in which a block task is implemented is usually defined via its decomposition into a sub-workflow. It is desirable that data elements available in the context of the undecomposed block task are available to all of the components that make up the corresponding sub-workflow. Similarly, it is useful if there is the ability to define new data elements within the context of the sub-workflow that can be utilised by each of the components during execution.

Figure 3 illustrates both of these scenarios, data element M is declared at the level of the block task C and is accessible both within the block task instance and throughout each of the task instances (X, Y and Z) in the corresponding sub-workflow. Similarly data element N is declared within the context of the sub-workflow itself and is available to all task instances in the sub-workflow. Depending on the underlying workflow system, it may also be accessible at the level of the corresponding block task.

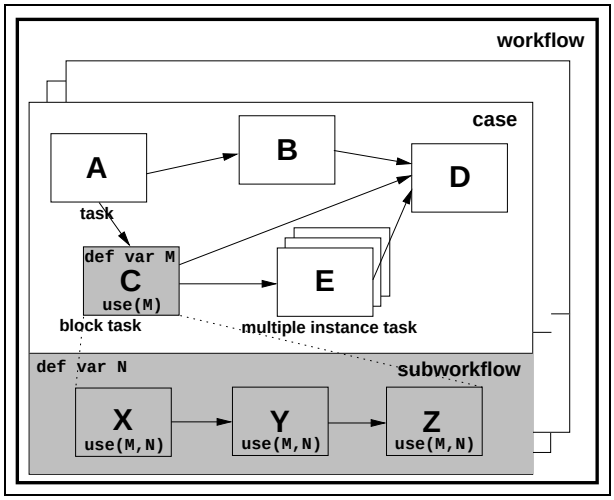

Figure 3: Block level data visibility

Implementation The concept of block data is widely supported by workflow systems and all but one of the offerings examined in this survey which supported the notion of sub-workflows[1] implemented it in some form. Staffware allows sub-workflows to specify their own data elements and also provides facilities for parent processes to pass data elements to sub-workflows as formal parameters. In MQSeries Workflow, sub-workflows can specify additional data elements in the data container that is used for passing data between task instances within the sub-workflow and restrict their scope to the sub-workflow. FLOWer and COSA also provide facilities for specifying data elements within the context of a sub-workflow.

Issues A major consideration in regard to block-structured tasks within a workflow is the handling of block data visibility where cascading block decompositions are supported and data elements are implicitly inherited by sub-workflows. As an example, in the preceding diagram block data sharing would enable a data element declared within the context of task C to be utilised by task X, but if X were also a block task would this data element also be accessible to task instances in the sub-workflow

[1] BPEL4WS which does not directly support sub-workflows is the only exception.

corresponding to X?

Solutions One approach to dealing with this issue adopted by workflow tools such as Staffware is to only allow one level of block data inheritance by default i.e. data elements declared in task instance C are implicitly available to X, Y and Z but not to further sub-workflow decompositions. Where further cascading of data elements is required, then this must be specifically catered for.

COSA allows a sub-workflow to access all data elements in a parent process and provides for arbitrary levels of cascading[2], however updates to data elements in sub-workflows are not automatically propagated back to the parent task.

Pattern 3 (Scope Data)

Description Data elements can be defined which are accessible by a subset of the tasks in a case.

Example

- The *initial tax estimate* variable is only used within the *Gather Return Data*, *Examine Similar Claims* and *Prepare Initial Return* tasks in the *Prepare Tax Return* process.

Motivation Where several tasks within a workflow coordinate their actions around a common data elements or set of data elements, it is useful to be able to define data elements that are bound to that subset of tasks in the overall workflow process.

Figure 4 illustrates the declaration of data element X which is scoped to tasks A, B and C. It can be freely accessed by these tasks but is not available to tasks D and E.

One of the major justifications for scopes in workflows is that they provide a means of binding data elements, error and compensation handlers to sets of related tasks within a case. This allows for more localised forms of recovery action to be undertaken in the event that errors or concurrency issues are detected.

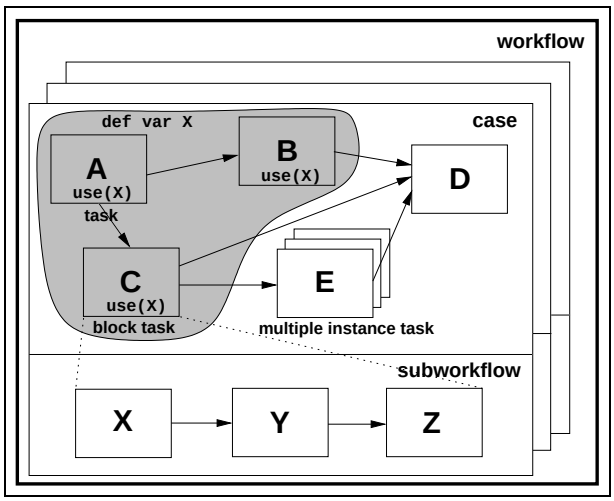

Figure 4: Scope level data visibility

[2]although more than four levels of nesting are not recommended.

Implementation The definition of scope data elements requires the ability to define the portion of the workflow process to which the data elements are bound. This is potentially difficult in workflows that are based on a graphical process notation but less difficult for those that utilise a textual definition format such as XML.

A significant distinction between scopes and blocks in a workflow context is that scopes provide a grouping mechanism within the same address space (or context) as the surrounding case elements. They do not define a new context and data passing to tasks within the scope does not rely on any specific data passing mechanisms other than normal task-to-task data transfer facilities.

BPEL4WS is the only offering examined that fully supports the notion of scope data elements. It provides support for a scope construct which allows related activities, variables and exception handlers to be logically grouped together. FLOWer supports 'restricted data elements' which can have their values set by nominated tasks although they are more widely readable.

Issues Potential exists for variables named within a scope to have the same name as a variable in the surrounding block in which the scope is defined.

Solutions The default handling for this BPEL4WS is that the innermost context in which a variable is defined indicates which variable should be used in any given situation. Variables within a given scope must be unique.

Pattern 4 (Multiple Instance Data)

Description Tasks which are able to execute multiple times within a single workflow case can define data elements which are specific to an individual execution instance.

Example

- Each instance of the *Expert Assessment* task is provided with the *case history* and *test results* at commencement and manages its own *working notes* until it returns a *verdict* at completion.

Motivation Where a task executes multiple times, it is useful to be able to define a set of data elements which are specific to each individual execution instance. The values of these elements may be passed to the task instance at commencement. Alternatively these data items may be used as working storage during its execution. There are three distinct scenarios in which a task could be executed more than once:

1. Where a particular task is designated as a multiple instance task and once it is enabled, multiple instances of it are initiated simultaneously.

2. Where distinct tasks in a workflow process share the same implementation.

3. Where a task can receive multiple initiation triggers (i.e. multiple tokens in a Petri-net sense) during the operation of a workflow case.

Each of these scenarios is illustrated in Figure 5 using the YAWL notation. Further details on YAWL can be found in [AH04].

Implementation The ability to support distinct data elements in multiple task instances presumes the workflow engine is also able to support data elements that can be bound specifically to individual tasks (i.e. pattern 1) in some form. Workflow engines lacking this capability are unable to facilitate the isolation of data elements

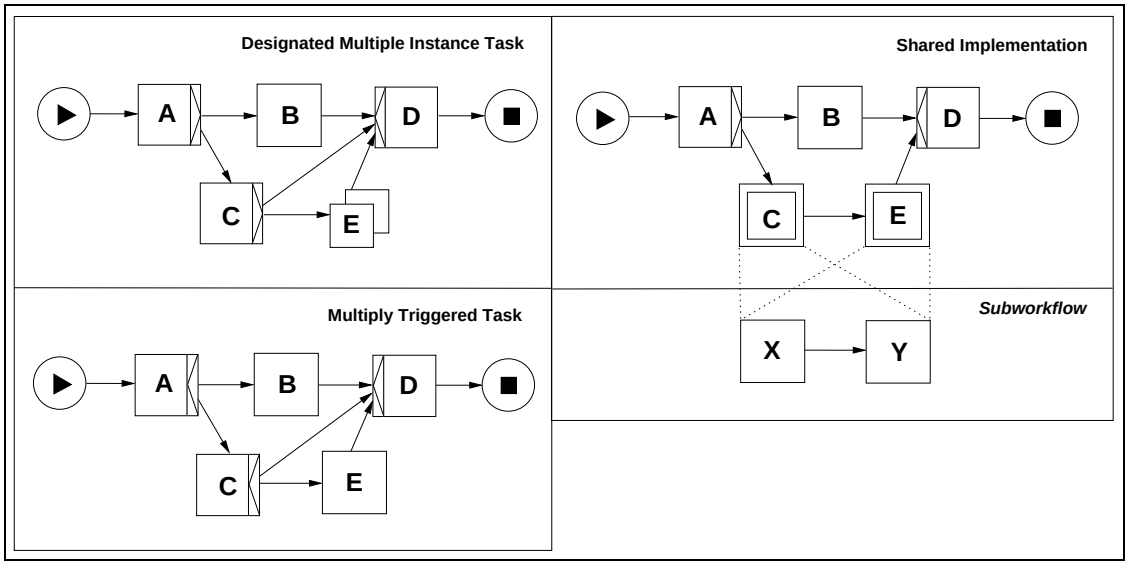

Figure 5: Alternative implementations of multiple instance tasks

between task instances for any of the scenarios identified above. In addition to this, there are also other prerequisites for individual scenarios as described below.

In order to support multiple instance data in the first of the scenarios identified above, the workflow engine must also support designated multiple instance tasks.

For the second scenario, it must be possible for two or more distinct block tasks to share the same implementation (i.e. the same underlying sub-workflow) and the workflow engine must support block-level data.

The third scenario requires the workflow engine to provide task-level data binding and support the capability for a given task to be able to receive multiple triggers. Each of the instances should have a mutually distinct set of data elements.

Of the various multiple instance scenarios identified, FLOWer provides support for the first of them[3]. MQSeries Workflow and COSA can support the second and MQSeries Workflow, COSA and XPDL directly support the third scenario. Staffware can potentially support the third scenario also, however programmatic extensions would be necessary to map individual instances to distinct case level variables.

Issues A significant issue that arises for workflow systems that support designated multiple instance tasks such as FLOWer involves the partitioning of composite data elements (such as an array) in a way that ensures each task instance receives a distinct portion of the data element and also that the entire data element is passed to one of the multiple task instances.

Solutions FLOWer has a unique means of addressing this problem through mapping objects which allow sections of composite data element in the form of an array (e.g. X[1], X[2] and so on) to be allocated to individual instances of a multiple instance task (known as a dynamic plan). Each multiple task instance only sees the element it has been allocated and each task instance has the same naming for each

[3]Recent documentation [GP03] suggests that Staffware will also support this construct shortly through addition of the Multiple Process Selection construct, although it has not been observed in the current release.

of these elements (i.e. X). At the conclusion of all of the multiple instances, the data elements are coalesced back into the composite form together with any changes that have been made.

Pattern 5 (Case Data)

Description Data elements are supported which are specific to a process instance or case of a workflow. They can be accessed by all components of the workflow during the execution of the case.

Example

- The *employee assessment results* can be accessed by all of the tasks during this execution instance of the *Performance Assessment* workflow.

Motivation Data elements defined at case level effectively provide global data storage during the execution of a specific case. Through their use, data can be made accessible to all workflow components without the need to explicitly denote the means by which it is passed between them.

Figure 6 illustrates the use of the case level data element X which is utilised by all of the tasks throughout a workflow case (including those in sub-workflows).

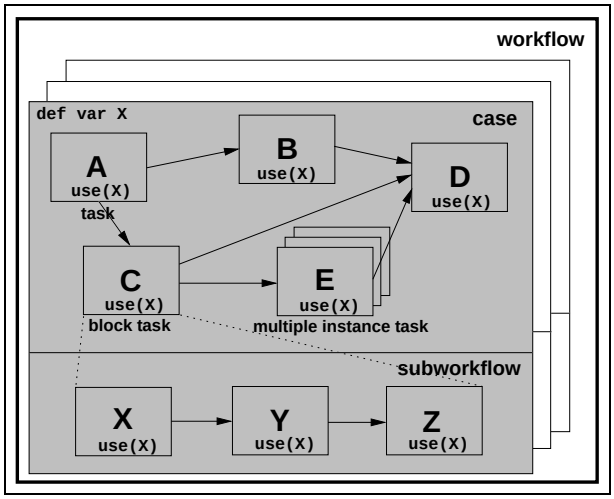

Figure 6: Case level data visibility

Implementation Most workflow engines support the notion of case data in some form, however the approaches to its implementation vary widely. Staffware implements a data management strategy that is based on the notion of a common data store for each workflow case although individual data fields must be explicitly passed to sub-workflows in order to make them accessible to the tasks within them. MQSeries Workflow takes a different approach with a global data store being defined for each workflow case for case level data elements but distinct data passing conventions needing to be specified to make this data accessible to individual tasks. COSA provides a series of tiers of data constructs with the INSTANCE tool agent corresponding to case level data elements which are globally accessible throughout a workflow case (and associated sub-workflows) by default. In FLOWer, XPDL and BPEL4WS, the

default binding for data elements is at case level and they are visible to all of the components in a process.

Issues One consideration that arises with the use of case level data is in ensuring that these elements are accessible, where required, to the components of a sub-workflow associated with a specific workflow case (e.g. as the definition of a block task).

Solutions In some workflow tools, sub-workflows that are linked to a workflow process do not seem to be considered to be part of the same execution context as the main workflow process. To remedy this issue, tools such as Staffware and MQSeries Workflow require that case level data elements be explicitly passed to and from sub-workflows as parameters in order to make them visible to the various components at this level. In COSA, FLOWer and XPDL, they are visible to all sub-workflows by default.

Pattern 6 (Workflow Data)

Description Data elements are supported which are accessible to all components in each and every case of the workflow and are within the control of the workflow system.

Example

- The *risk/premium matrix* can be utilised by all of the cases of the *Write Insurance Policy* workflow and all tasks within each case.
- During the course of workflow execution, a number of cases will be selected for auditing each hour. The number selected will be based on the *average case execution time* for the previous six hours.

Motivation Some data elements have sufficiently broad applicability that it is desirable to make them accessible to every component in all cases of workflow execution. Data that falls into this category includes startup parameters to the workflow engine, global application data that is frequently used and production information that governs the potential course of execution that each workflow case may take.

Figure 7 illustrates the extent of workflow data visibility. Note that in contrast to case level data elements which are typically only visible to tasks in the main workflow case in which they are declared, workflow-level data elements are visible globally throughout all workflow cases - both to the main body of the case and also to sub-workflows linked to it.

Implementation In order to make data elements broadly accessible to all workflow cases, most workflow engines address this requirement by utilising persistent storage, typically in the form of a database. This may be provided directly as an internal facility by the workflow engine (tables, persistent lists, DEFINITION tool agents and packages in the case of Staffware, MQSeries Workflow, COSA and XPDL respectively) or by linking in or providing access facilities to a suitable third-party product.

Issues The main issue associated with workflow-level data is in managing concurrent access to it by multiple processes.

Solutions Concurrency management is not an area that is currently well-addressed by commercial workflow engines. Where a workflow engine provides for workflow-level data, the general solutions to the concurrency problem tend to be either to allow for

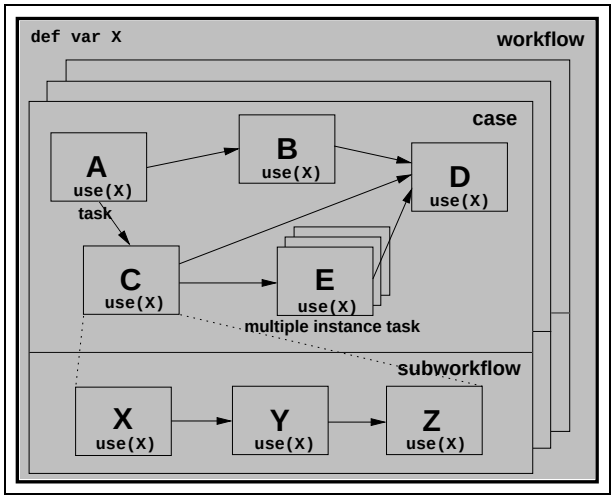

Figure 7: Workflow data visibility

the use of a third party transaction manager (e.g. Staffware which allows Tuxedo to be used for transaction management) or to leave the problem to the auspices of the workflow developer (e.g. COSA).

There has been significant research interest over recent years in the various forms of advanced transaction support that are required for business processes [Gre02, RS95, AAA+96, WS97, DHL01] and several prototypes have been constructed which provide varying degrees of concurrency management and transactional support for workflow systems e.g. METEOR [KS95], EXOTICA [AAA+96], WIDE [GVA01], CrossFlow [VDGK00] and ConTracts [WR92], although most of these advances have yet to be incorporated in mainsteam commercial products.

Pattern 7 (Environment Data)

Description Data elements which exist in the external operating environment are able to be accessed by components of the workflow during execution.

Example

- Where required, tasks in the *System Monitoring* workflow can access the *temperature sensor data* from the operating environment.

Motivation Direct access to environmentally managed data by workflow tasks or cases during execution can significantly simplify workflow processes and improve their ability to respond to changes in the broader operational context. External data may be sourced from a variety of distinct locations including external databases, applications that are currently executing or can be initiated in the operating environment and services that mediate access to various data repositories and distribution mechanisms e.g stock price feeds. These scenarios are illustrated in Figure 8.

Implementation The ability to access external data elements generally requires the ability to connect to an interface or interprocess communication (IPC) facility in the operating environment or to invoke an external service which will supply data elements. Facilities for achieving this may be either *explicitly* or *implicitly* supported by the workflow engine.

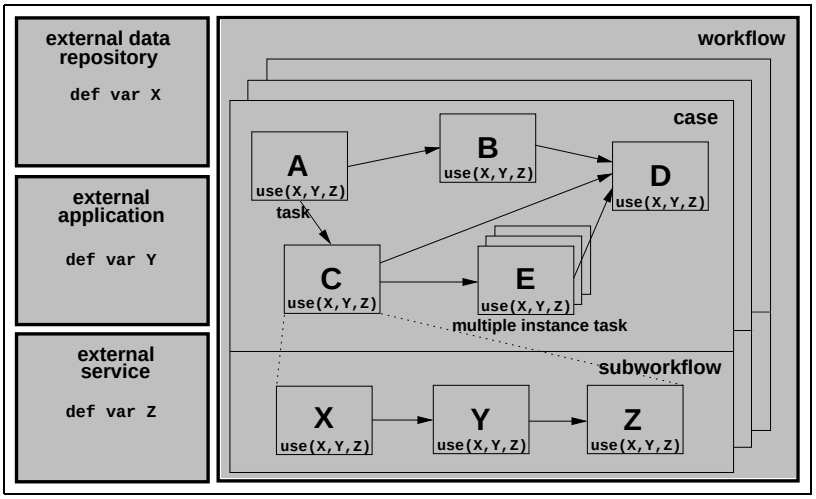

Figure 8: Environment data visibility

Explicit support involves the direct provision by the workflow engine of constructs for accessing external data sources. Typically these take the form of specific elements that can be included in the design-time workflow model. Staffware provides the Automatic Step construct as well as script commands which enable specific items of data to be requested from an external applications. COSA provides the Tool Agent interfaces which provide a number of facilities for accessing external data elements. FLOWer implements Mapping Objects which allow data elements to be copied from external databases into the workflow engine, updated as required with case data and copied back to the underlying database. It also allows text files to be utilised in workflow actions and has a series of interfaces for external database integration. BPEL4WS provide facilities that enable external web services to be invoked.

Implicit support occurs in workflow engines such as MQSeries Workflow where the actual implementation of individual workflow tasks is achieved by the development of associated programs in a procedural language such as C++ or Java. In this situation, access to external data occurs within individual task implementations by extending the program code to incorporate the required integration capabilities.

Issues There are a multitude of ways in which external data elements can be utilised within a workflow system. It is infeasible for any workflow tool to support more than a handful of them. This raises the issue of the minimal set of external integration facilities required for effective external data integration.

Solutions There is no definitive answer to this problem as the set of facilities required depends on the context in which the tool will be utilised. For the purposes of this research however, we consider it sufficient if a tool can demonstrate the ability to access data files (in text or binary format) in the operating environment and is able to access an external API (e.g. an XML, DDE or ODBC interface) through which data requests can be dynamically framed.

2.3 Data interaction

In this section, we examine the various ways in which data elements can be passed between components in a workflow process and how the characteristics of the individual components can influence the manner in which the trafficking of data elements occurs. Of particular interest is the distinction between the communication of data between components within a workflow engine as against the data-oriented interaction of a workflow element with the external environment.

2.3.1 Internal data interaction

Pattern 8 (Data Interaction – Task to Task)

Description The ability to communicate data elements between one task instance and another within the same case.

Example

- The *Determine Fuel Consumption* task requires the coordinates determined by the *Identify Shortest Route* task before it can proceed.

Motivation The passing of data elements between tasks is a fundamental aspect of workflow systems. In many situations, individual tasks execute in their own distinct address space and do not share significant amounts of data on a global basis. This necessitates the ability to move commonly used data elements between distinct tasks as required.

Implementation All workflow engines support the notion of passing parameters from one task to another however, this may occur in a number of distinct ways depending on the relationship between the data perspective and control flow perspective within the workflow.

There are three main approaches as illustrated in Figure 9.

The distinctions between each of these are as follows:

- *Integrated control and data channels* – where both control flow and data are passed simultaneously between tasks utilising the same channel. In the example, task B receives the data elements X and Y at exactly the same time that control is passed to it. Whilst conceptually simple, one of the disadvantages of this approach to data passing is that it requires all data elements that may be used some time later in the workflow process to be passed with the thread of control regardless of whether the next task will use them or not. E.g. task B does not use data element Y but it is passed to it because task C will subsequently require access to it.

- *Distinct data channels* – in which data is passed between workflow tasks via explicit data channels which are distinct from the process control links within the workflow design. Under this approach, the coordination of data and control passing is usually not specifically identified. It is generally assumed that when control is passed to a task that has incoming data channels, the data elements specified on these channels will be available at the time of task commencement.

- *No data passing* – where tasks share the same data elements (typically via access to globally shared data) and no explicit data passing is required (cf. patterns

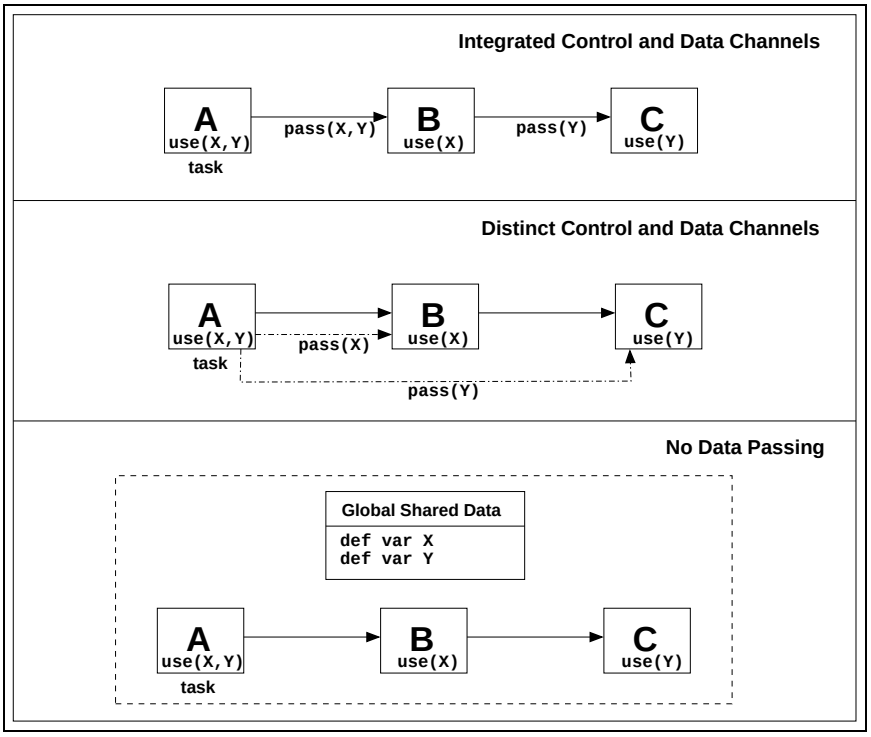

Figure 9: Approaches to data interaction between tasks

4, 5 and 6). This approach to data sharing is based on tasks having shared *a priori* knowledge of the naming and location of common data elements. It also assumes that the implementation is able to deal with potential concurrency issues that may arise where several task instances seek to access the same data element.

The majority of offerings examined adopt the third strategy. Staffware, FLOWer, COSA and XPDL all facilitate the passing of data through case-level data repositories accessible by all tasks. BPEL4WS utilises a combination of the first and third approaches. Variables can be bound to scopes within a process definition which may encompass a number of tasks, but there is also the ability for messages to be passed between tasks when control passes from one task to another. MQSeries Workflow adopts the second mechanism with data elements being passed between tasks in the form of data containers via distinct data channels.

Issues Where there is no data passing between tasks and a common data store is utilised by several tasks for communicating data elements, there is the potential for concurrency problems to arise, particularly if the case involves parallel execution paths. This may lead to inconsistent results depending on the task execution sequence that is taken.

Solutions Concurrency control is handled in a variety of different ways by the offerings examined in Section 5. FLOWer avoids the problem by only allowing one active user or process that can update data elements in a case at any time (although other processes and users can access data elements for reading). BPEL4WS supports serialisable scopes which allow compensation handlers to be defined for groups of tasks

that access the same data elements. A compensation handler is a procedure that aims to undo or compensate for the effects of the failure of a task on other tasks that may rely on it or on data that it has affected. Staffware provides the option to utilise an external transaction manager (Tuxedo) within the context of the workflow cases that it facilitates.

Pattern 9 (Data Interaction – Block Task to Sub-Workflow Decomposition)
Description The ability to pass data elements from a block task instance to the corresponding sub-workflow that defines its implementation.
Example

- *Customer claims* data is passed to the *Calculate Likely Tax Return* block task whose implementation is defined by a series of tasks in the form of a sub-workflow. The *customer claims* data is passed to the sub-workflow and is visible to each of these sub-tasks.

Motivation Most workflow systems support the notion of composite or block tasks in some form. These are analogous to the programmatic notion of procedure calls and indicate a task whose implementation is described in further detail at another level of abstraction (typically elsewhere in the workflow design) utilising using the same range of workflow constructs.

The question that arises when data is passed to a block element is whether it is immediately accessible by all of the tasks that define its actual implementation or if some form of explicit data passing must occur between the block task and the sub-workflow.
Implementation Typically one of three approaches is taken to handling the communication of parameters from a block task to the underlying implementation. Each of these is illustrated in Figure 10. The characteristics of each approach are as follows:

- *Implicit data passing* – data passed to the block task is immediately accessible to all sub-tasks which make up the underlying implementation. In effect the main block task and the corresponding sub-workflow share the same address space and no explicit data passing is necessary.

- *Explicit data passing via parameters* – data elements supplied to the block task must be specifically passed as parameters to the underlying sub-workflow implementation. The second example in Figure 10 illustrates how the specification of parameters can handle the mapping of data elements at block task level to data elements at sub-workflow level with distinct names. This capability provides a degree of independence in the naming of data elements between the block task and sub-workflow implementation and is particularly important in the situation where several block tasks share the same sub-workflow implementation.

- *Explicit data passing via data channels* – data elements supplied to the block task are specifically passed via data channels to all tasks in the sub-workflow that require access to them.

It is important to note that the first approach does not involve any actual data passing between block activity and implementation, rather the block level data elements are made accessible to the sub-workflow. This strategy is adopted by FLOWer

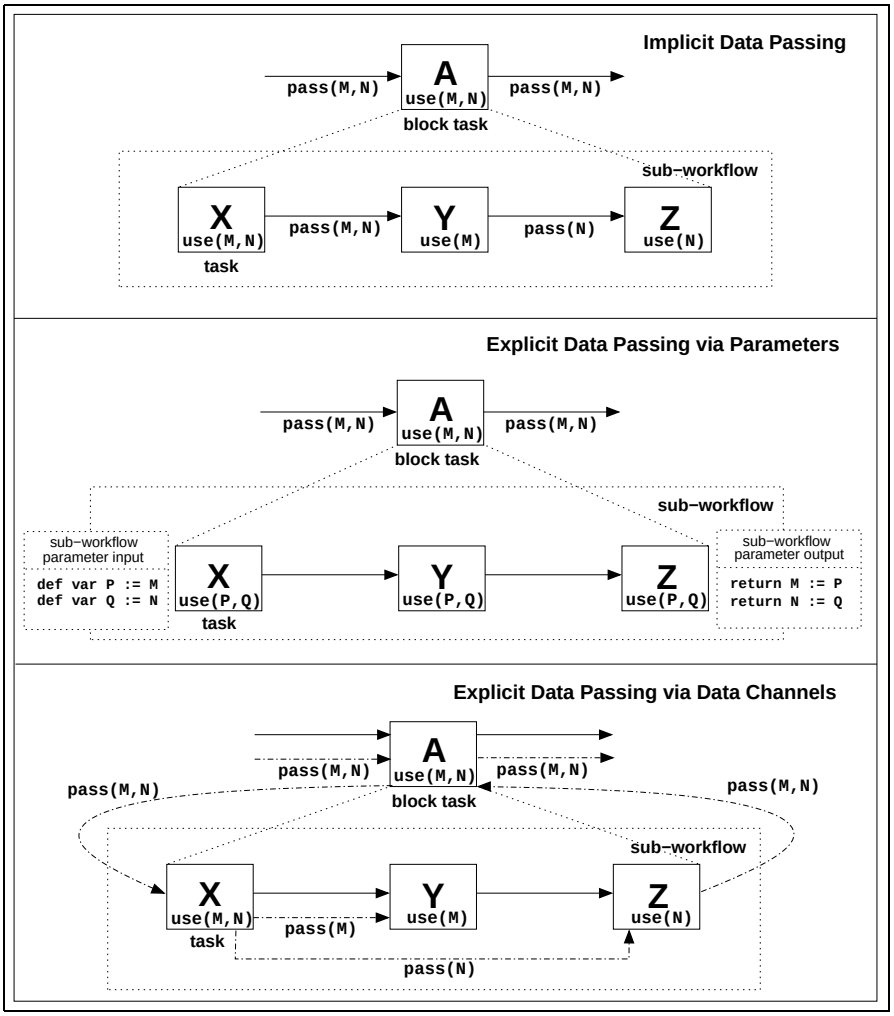

Figure 10: Approaches to data interaction from block tasks to corresponding sub-workflows

and COSA for data passing to sub-workflows. In both cases, the sub-workflow is presented with the entire range of data elements utilised by the block task and there is no opportunity to limit the scope of items passed.

The third approach relies on the passing of data elements between tasks to communicate between block task and sub-workflow. Both Staffware and XPDL utilise this mechanism for passing data elements to sub-workflows.

In contrast, the second approach necessitates the creation of new data elements at sub-workflow level to accept the incoming data values from the block activity. MQSeries Workflow follows this strategy and sink and source nodes are used to pass data containers between the parent task and corresponding sub-workflow.

Issues One consideration that may arise where the explicit parameter passing approach is adopted is whether the data elements in the block task and the sub-workflow are independent of each other (i.e. whether they exist in independent address spaces). If they do not, then the potential exists for concurrency issues to arise as tasks executing in parallel update the same data element.

Solutions This issue can arise with Staffware where sub-workflow data elements are passed as fields rather than parameters. The resolution to this issue is to map the input fields to fields with distinct names not used elsewhere during execution of the case.

Pattern 10 (Data Interaction – Sub-Workflow Decomposition to Block Task)

Description The ability to pass data elements from the underlying sub-workflow back to the corresponding block task instance.

Example

- The *Determine Optimal Trajectory* sub-workflow passes the coordinates of the *launch* and *target locations* and *flight plan* back to the block task.

Motivation At the conclusion of the underlying sub-workflow, the data elements which have resulted from its execution need to be made available to the block activity which called it.

Implementation The approaches taken to handling this pattern are essentially the same as those identified for pattern 9. Where data elements are passed on an explicit basis, a mapping needs to be specified for each output parameter indicating which data element at block level will receive the relevant output value.

Issues One difficulty that can arise with the implementation of this pattern occurs when there is not a strict correspondence between the data elements returned from the sub-workflow and the receiving data elements at block task level. E.g. the sub-workflow returns more data elements than the block task is expecting, possibly as a result of additional data elements being created during its execution.

Solutions This problem can be solved in one of two ways. Some workflow engines such as Staffware support the ability for block tasks to create data elements at block task level for those data items at sub-workflow level which it has not previously encountered. Other products such as MQSeries Workflow require a strict mapping to be defined between sub-workflow and block task data elements to prevent this situation from arising.

Pattern 11 (Data Interaction – to Multiple Instance Task)

Description The ability to pass data elements from a preceding task instance to a subsequent task which is able to support multiple execution instances. This may involve passing the data elements to all instances of the multiple instance task or distributing them on a selective basis.

Examples

- The *Identify Witnesses* task passes the *witness list* to the *Interview Witnesses* task. This data is available to each instance of the *Interview Witnesses* task at commencement.

- The *New Albums List* is passed to the *Review Album* task and one task instance is started for each entry on the list. At commencement, each of the *Review Album* task instances is allocated a distinct entry from the *New Albums List* to review.

Motivation Where a task is capable of being invoked multiple times, a means is required of controlling which data elements are passed to each of the execution instances. This may involve ensuring that each task instance receives all of the data elements passed to it (possibly on a shared basis) or distributing the data elements across each of the execution instances on some predefined basis.

Implementation There are three potential approaches to passing data elements to multiple instance tasks as illustrated in Figure 11.

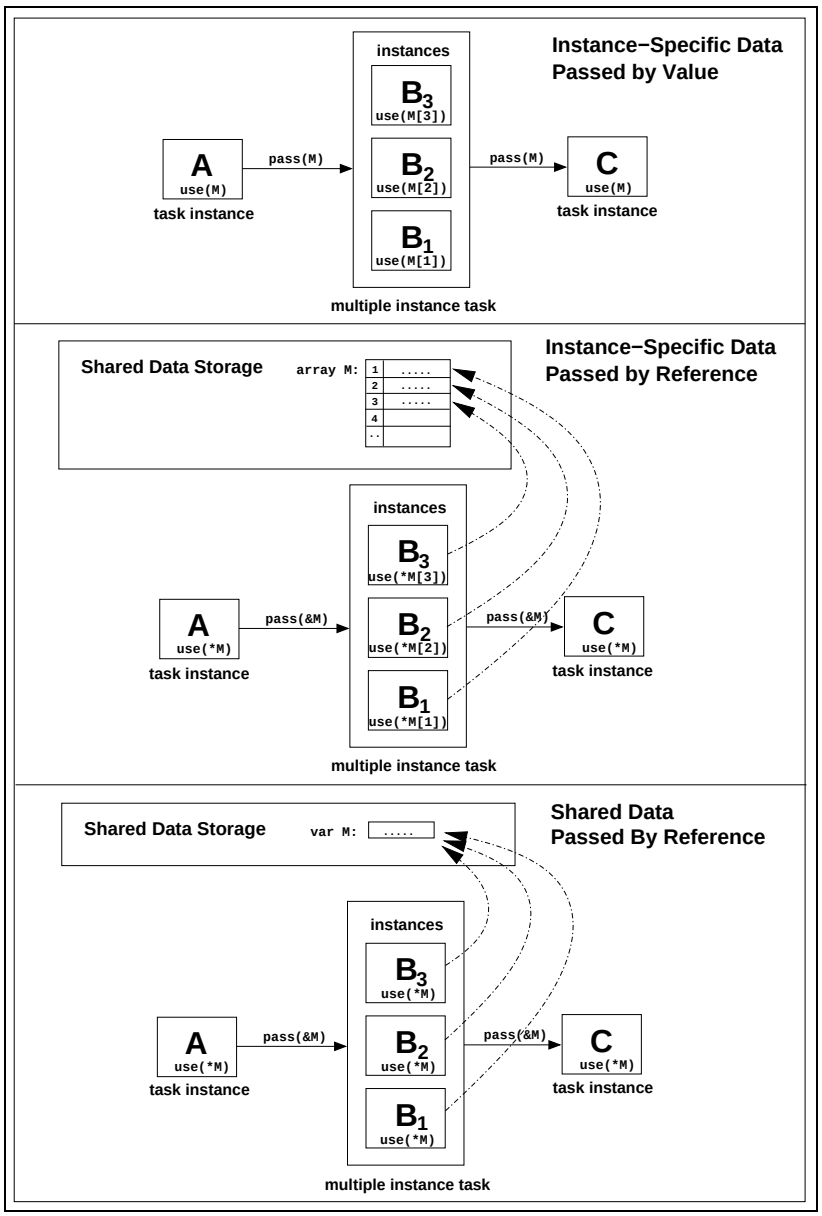

Figure 11: Data interaction approaches for multiple instance tasks

As a general rule, it is possible either to pass a data element to all task instances or to distribute one item from it (assuming it is a composite data element such as an array or a set) to each task instance. Indeed the number of task instances that

are initiated may be based on the number of individual items in the composite data element.

The specifics of each of these approaches are discussed below.

- *Instance-specific data passed by value* – this involves the distribution of a data element passed by value to task instances on the basis of one item of the data element per task instance (in the example shown, task instance B_1 receives M[1], B_2 receives M[2] and so on). As the data element is passed by value, each task instance receives a copy of the item passed to it in its own address space. At the conclusion of each of the task instances, the data element is reassembled from the distributed items and passed to the subsequent task instance.

- *Instance-specific data passed by reference* – this scenario is similar to that described above except that the task instances are passed a reference to a specific item in the data element rather than the value of the item. This approach obviates the need to reassemble the data element at the conclusion of the task instances.

- *Shared data passed by reference* – in this situation all task instances are passed a reference to the same data element. Whilst this allows all task instances to access the same data element, it does not address the issue of concurrency control should one of the task instances amend the value of the data element (or indeed if it is altered by some other component of the workflow).

FLOWer provides facilities for instance-specific data to be passed by reference whereby an array can be passed to a designated multiple instance task and specific sub-components of it can be mapped to individual task instances. It also allows for shared data elements to be passed by reference to all task instances.

Issues Where a task is able to execute multiple times but not all instances are created at the same point, an issue that arises is whether the values of data elements are set for all execution instances at the time at which the multiple instance task is initiated or whether they can be fixed after this occurs but prior to the actual invocation of the task instance to which they relate.

Solutions In FLOWer, the Dynamic Plan construct allows the data for individual task instances to be specified at any time prior to the actual invocation of the task. The passing of data elements to specific task instances is handled via Mapping Array data structures. These can be extended at any time during the execution of a Dynamic Plan, allowing for new task instances to be created 'on the fly' and the data corresponding to them to be specified at the latest possible time.

Pattern 12 (Data Interaction – from Multiple Instance Task)

Description The ability to pass data elements from a task which supports multiple execution instances to a subsequent task.

Example

- At the conclusion of the various instances of the *Record Lap Time* task, the *list of lap times* is passed to the *Prepare Grid* task.

Motivation Each execution instance of a multiple instance task effectively operates independently from other instances and as such, has a requirement to pass on data elements at its conclusion to subsequent tasks.

Implementation In general, data is passed from a multiple instance task to subsequent tasks when a certain number of the task instances have concluded. The various scenarios in which this may occur are illustrated in Figure 11. These usually coincide with the passing of control from the multiple instance task although data and control flow do not necessarily need to be fully integrated. In the case where data elements are passed by reference, the location rather than the values are passed on at task completion (obviously this implies that the data values may be accessed by other components of the workflow prior to completion of the multiple instance task as they reside in shared storage).

Issues One issue that arises with multiple instance tasks is the point at which the output data elements from them are available (in some aggregate form) to subsequent tasks.

Solutions In the case of FLOWer, all instances must be complete before the output data elements are available to subsequent tasks. There is no notion of partial access to data elements from task instances that have completed at a given point in time.

Pattern 13 (Data Interaction – Case to Case)

Description The passing of data elements from one case of a workflow during its execution to another case that is executing concurrently.

Example

- During execution of a case of the *Re-balance Portfolio* workflow the *best price* identified for each security is passed to subsequent cases.

Motivation Where the work completed and the results obtained during the course of one workflow case are likely to be of use to subsequent cases, a means of communicating them to both currently executing and subsequent cases is required.

Implementation Direct support for this function requires the availability of a function within the workflow tool that supports a given case initiating the communication (and possibly updating) of data elements with a nominated workflow case.

Alternatively, it is possible to achieve the same outcome indirectly by writing them back to a shared store of persistent data known to both the initiating and receiving cases. This may be either an internally maintained facility at workflow level or an external data repository[4]. Both of these scenarios are illustrated in Figure 12.

Each of these approaches require the communicating cases to have *a priori* knowledge of the location of the data element that is to be used for data passing. They also require the availability of a solution to address the potential concurrency issues that may arise where multiple workflow cases wish to communicate with each other.

None of the offerings examined supported a direct method for communicating data elements between workflow cases however it is possible to achieve the same result indirectly for each of them using the methods described above.

[4] Although this does not constitute 'true' case to case data passing, it indirectly achieves the same outcome.

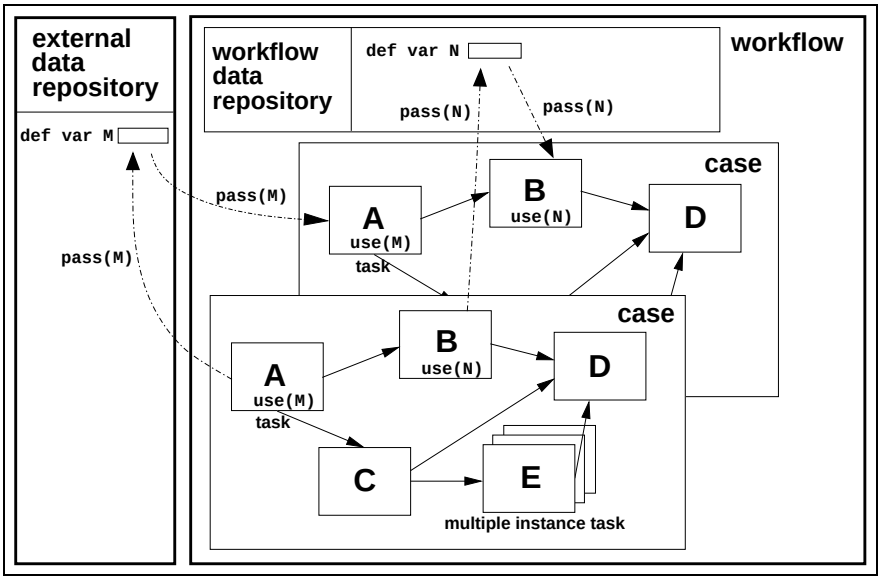

Figure 12: Data interaction between workflow cases

Issues The main consideration associated with this pattern is in establishing an effective means of linking related workflow cases and their associated data together in a meaningful way.

Solutions None of the workflow engines examined address this need, however there are offerings that provide solutions to the issue. In Vectus[5] it is possible to relate cases. There are four types of relationships "parent", "child", "sibling", and "master". These relationships can be used to navigate through cases at runtime. For example, one case can schedule tasks in another flow, terminate a task in another flow, create new cases, etc. It is also possible to share data using a dot notation similar to Object Constraint Language [OMG03]. The "master" relation can be used to create a proxy shared among related cases to show all tasks related to these cases. A similar construct is also present in the BizAgi[6] product.

2.3.2 External data passing

External data passing involves the communication of data between a component of a workflow process and some form of information resource or service that is operated outside of the context of the workflow engine. The notion of being external to the context of the workflow engine applies not only in technology terms but also implies that the operation of the external service or resource is independent of that of the workflow engine. The various types of interaction between elements within a workflow system and the broader operational environment are considered in this section.

Pattern 14 (Data Interaction – Task to Environment – Push-Oriented)
Description The ability of a task to initiate the passing of data elements to a resource or service in the operating environment.

[5] `http://www.london-bridge.com`
[6] `http://www.visionsoftware.biz`

Example

- The *Calculate Mileage* task passes the *mileage data* to the *Corporate Logbook* database for permanent recording.

Motivation The passing of data from a workflow task to an external resource is most likely to be initiated by the task itself during its execution. Depending on the specific requirements of the data passing interaction, it may connect to an existing API or service interface in the operating environment or it may actually invoke the application or service to which it is forwarding the data.

Figure 13 illustrates the various data passing scenarios between workflow tasks and the operating environment.

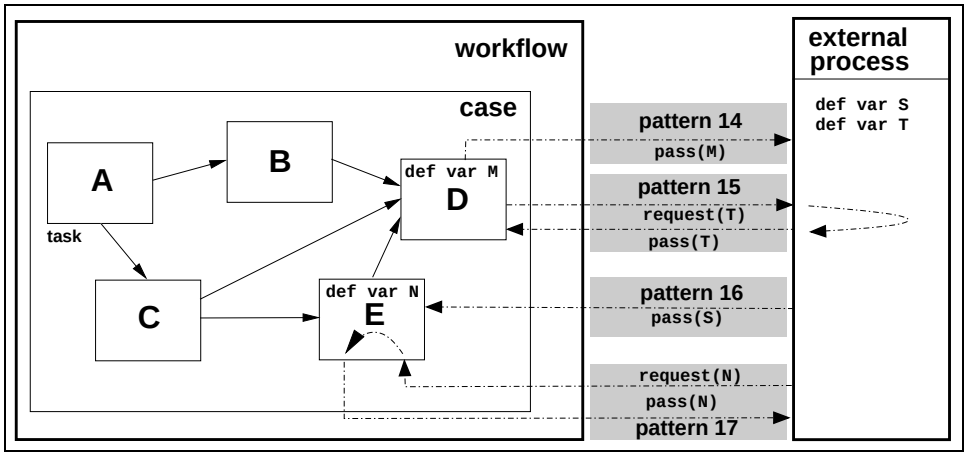

Figure 13: Data interaction between workflow tasks and the operating environment

Implementation There are a wide range of ways in which this pattern is achieved in workflow engines, however these tend to divide into two categories:

- *Explicit integration mechanisms* – where the workflow system provides specific constructs for passing data to the external environment.
- *Implicit integration mechanisms* – where the data passing occurs implicitly within the programmatic implementations that make up tasks in the workflow process and is not directly supported by the workflow environment.

In both cases, the data passing activity is initiated by the task and occurs synchronously with task execution. Although, it is typically task-level data elements that are trafficked, any data items that are accessible to the task (i.e. parameters, case and workflow data) may be transferred. Staffware provides the Automatic Step construct for passing data elements from task instances to external programs. FLOWer enables data to be passed to external applications using the Operation Action construct and to external databases using Mapping Objects. COSA provides a broad number of Tool Agents which facilitate the passing of data to external applications and databases. MQSeries Workflow is more limited in its support and delegates the passing of data elements to user-defined Program implementations. XPDL and BPEL4WS are limited to passing data elements to web services.

Issues One difficulty that can arise when sending data to an external application is knowing whether it was successfully delivered. This is particularly a problem when the external application does not provide immediate feedback on delivery status.

Solutions The most general solution to this problem is for a subsequent task in the case to request an update on the status of the delivery using a construct which conforms to pattern 15 and requires an answer to be provided by the external application. Alternatively, the external application can lodge a notification of the delivery using some form of event (i.e. pattern 37) or by passing data back to the workflow (i.e. pattern 16). This is analogous to the messaging notion of asynchronous callback in a workflow context.

Pattern 15 (Data Interaction – Environment to Task – Pull-Oriented)

Description The ability of a workflow task to request data elements from resources or services in the operational environment.

Example

- The *Determine Cost* task must request *cattle price* data from the *Cattle Market System* before it can proceed.

Motivation Workflow tasks require the means to proactively seek the latest information from known data sources in the operating environment during their execution. This may involve accessing the data from a known repository or invoking an external service in order to gain access to the required data elements.

Implementation Similar to pattern 14, distinct workflow engines support this pattern in a variety of ways however these approaches divide into two categories:

- *Explicit integration mechanisms* – where the workflow system provides specific constructs for accessing data in the external environment.
- *Implicit integration mechanisms* – where access to external data occurs at the level of the programmatic implementations that make up tasks in the workflow process and is not directly supported by the workflow engine. Interaction with external data sources typically utilises interprocess communication (IPC) facilities provided by the operating system facilities such as message queues or remote procedure calls, or enterprise application integration (EAI) mechanisms such as DCOM[7], CORBA[8] or JMS[9].

Staffware provides two distinct constructs that support this objective. Automatic Steps allow external systems to be called (e.g. databases or enterprise applications) and specific data items to be requested. Scripts allow external programs to be called either directly at system level or via system interfaces such as DDE[10] to access required data elements. FLOWer utilises Mapping Objects to extract data elements from external databases. COSA has a number of Tool Agent facilities for requesting data from external applications. XPDL and BPEL4WS provide facilities for the synchronous request of data from other web services.

[7] Distributed Component Object Model. See `http://www.microsoft.com` for details.
[8] Common Object Request Broker Architecture. See `http://www.omg.org` for details.
[9] Java Messaging Service. See `http://java.sun.com` for details.
[10] Dynamic Data Exchange. See `http://www.microsoft.com` for details.

In contrast, MQSeries Workflow does not provide any facilities for external integration and requires the underlying programs that implement workflow tasks to provide these capabilities where they are required.

Issues One difficulty with this style of interaction is that it can block progress of the requesting case if the external application has a long delivery time for the required information or is temporarily unavailable.

Solutions The only potential solution to this problem is for the requesting case not to wait for the requested data (or continue execution after a nominated timeout) and to implement some form of asynchronous notification of the required information (possibly along the lines of pattern 16). The disadvantage of this approach is that it complicates the overall interaction by requiring the external application to return the required information via an alternate path and necessitating the workflow to provide notification facilities.

Pattern 16 (Data Interaction – Environment to Task – Push-Oriented)

Description The ability for a workflow task to receive and utilise data elements passed to it from services and resources in the operating environment on an unscheduled basis.

Example

- During execution, the *Review Performance* task may receive new *engine telemetry* data from the *Wireless Car Sensors*.

Motivation An ongoing difficulty for workflow tasks is establishing mechanisms that enable them to be provided with new items of data as they become available from sources outside of the workflow system. This is particularly important in areas of volatile information where updates to existing data elements may occur frequently but not on a scheduled basis e.g. price updates for equities on the stock market. This pattern relates to the ability of tasks to receive new items of data as they become available without needing to proactively request them from external sources or suspend execution until updates arrive.

Implementation As for patterns 14 and 15, approaches to this pattern can be divided into *explicit* and *implicit* mechanisms. The main difficulty that arises in its implementation is in providing external processes with the addressing information that enables the routing of data elements to a specific task instance. Potential solutions to this include the provision of externally accessible execution monitors by workflow engines which indicate the identity of currently executing tasks or task instances recording their identity with a shared registry service in the operating environment.

An additional consideration is the ability of workflow tasks to be able to asynchronously wait for and respond to data passing activities without impacting the actual progress of the task. This necessitates the availability of asynchronous communication facilities at task level – either provided as some form of (*explicit*) task construct by the workflow engine or able to be (*implicitly*) included in the programmatic implementations of tasks.

A number of the offerings examined support this pattern in some form. COSA provides the ability to pass data elements to tasks via the trigger construct. BPEL4WS provides a similar capability with the receive construct and event handlers. FLOWer allows the value of data elements associated with task forms to be updated via the

chp_frm_setfield API call. In Staffware, this pattern can be indirectly achieved by using the Event Step construct which allows external processes to update field data during the execution of a workflow case. However, a 'dummy' task is required to service the query request.

Issues The major difficulty associated with this pattern is in providing a means for external applications to identify the specific task instance in which they wish to update a data element.

Solutions In general the solution to this problem requires the external application to have knowledge of both the case instance and the specific task in which the data element resides. Details of currently executing cases can only be determined with reference to the workflow engine and most offerings provide a facility or API call to support this requirement. The external application will most likely require a priori knowledge of the identity of the task in which the data element resides.

Pattern 17 (Data Interaction – Task to Environment – Pull-Oriented)

Description The ability of a workflow task to receive and respond to requests for data elements from services and resources in the operational environment.

Example

- During execution, the *Assess Quality* task may receive requests for current data from the *Quality Audit* web service handler. It must provide this service with details of all current data elements.

Motivation In some cases, the requests for access to task instance data are initiated by processes outside the workflow environment. These requests need to be handled as soon as they are received but should be processed in a manner that minimises any potential impact on the task instance from which they are sourcing data.

Implementation The ability to access data from a task instance can be handled in one of three ways:

1. The workflow engine can provide a means of accessing task instance data from the external environment.

2. During their execution, tasks instances publish data values to a well-known location e.g. a database that can be accessed externally.

3. Task instances incorporate facilities to service requests for their data from external processes.

In practice, this facility is not widely supported as an explicit construct by the offerings examined. BPEL4WS provides direct support for the pattern (via the receive construct and event handlers). Staffware provides the EIS Report construct and EIS Case Data Extraction facilities to enable the values of data elements to be requested from workflow cases.

FLOWer and COSA provide indirect solutions via the chp_frm_getfield API call and trigger/tool agent mechanisms respectively although both methods have limited generality.

Issues Similar difficulties exist with the utilisation of this pattern to those identified above for pattern 16.

Solutions As detailed above for pattern 16.

Pattern 18 (Data Interaction – Case to Environment – Push-Oriented)

Description The ability of a workflow case to initiate the passing of data elements to a resource or service in the operational environment.

Example

- At its conclusion, each case of the *Perform Reconciliation* workflow passes its results to the *Corporate Audit* database for permanent storage.

Motivation An alternative (or possible extension) to task-level data passing is to provide facilities for passing data at the level of the workflow case. The various options for this approach to data passing are illustrated in Figure 14.

This pattern is analogous to pattern 14 except that it operates in the context of a process instance.

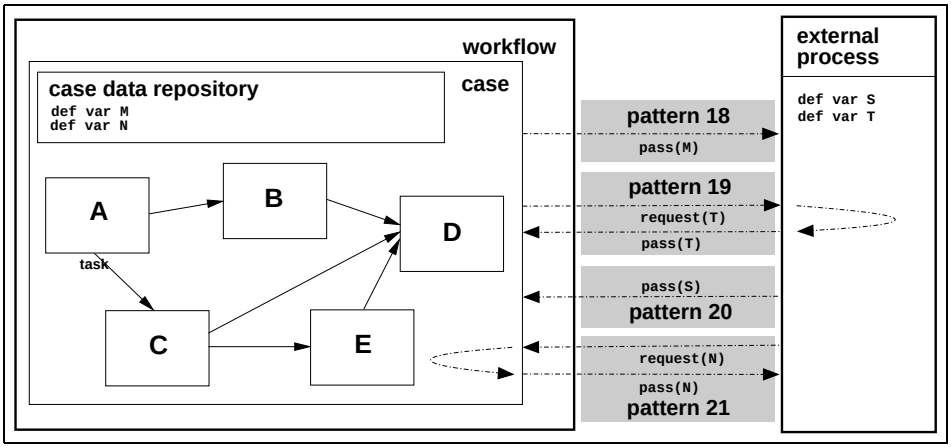

Figure 14: Data interaction between cases and the operating environment

Implementation This pattern has not been widely observed in the tools evaluated in Section 5. Its implementation requires explicit support by the workflow engine for case level data passing which is independent of the task instances that comprise the case.

Maximal flexibility for this pattern is gained through the use of a rule-based approach to the triggering of the data passing action. Potential invocation criteria include state-based conditions (e.g. case initiation, case completion), data-based conditions (e.g. pass the data element when a nominated data condition is satisfied) and temporal conditions (e.g. emit the value of a data element periodically during case execution).

FLOWer provides a mechanism for synchronising case data elements with an external database through the use of mapping constructs which are triggered for each activity in a plan. This provides a mechanism for providing external visibility of data elements during the execution of a case.

Issues None observed.

Solutions N/A.

Pattern 19 (Data Interaction – Environment to Case – Pull-Oriented)

Description The ability of a workflow case to request data from services or resources in the operational environment.

Example

- At any time cases of the *Process Suspect* workflow may request additional data from the *Police Records System*.

Motivation In the event that a workflow case requires access to the most current values of external data elements during execution, it may not be sufficient for these values to be specified at the initiation of the case. Instead, facilities may be required for them to be accessed during the course of execution at the point in the case where they are most likely to be needed.

Implementation Similar to pattern 17, this pattern has not been widely observed in the tools evaluated in Section 5. Once again, its implementation requires explicit support by the workflow engine for the extraction of data elements from external applications.

FLOWer provides this capability via mapping functions linked to each activity in a plan which extract required data elements from an external database.

Issues None observed.

Solutions N/A.

Pattern 20 (Data Interaction – Environment to Case – Push-Oriented)

Description The ability of a workflow case to accept data elements passed to it from services or resources in the operating environment.

Example

- During execution, each case of the *Evaluate Fitness* workflow may receive additional *fitness measurements* data from the *Biometry system*.

Motivation During the course of the execution of a workflow case, new case-level data elements may become available or the values of existing items may change. A means of communicating these data updates to the workflow case is required.

Implementation There are two distinct mechanisms by which this pattern can be achieved:

- Values for data elements can be specified at the initiation of a specific case.
- The workflow can provide facilities for enabling update of data elements during the execution of a case.

Staffware supports the former approach allowing startup values to be specified for cases. MQSeries Workflow also allows values to be specified for data elements at case commencement. FLOWer provides direct support for update of data elements during case execution through the chp_dat_setval API call.

Issues None observed.

Solutions N/A.

Pattern 21 (Data Interaction – Case to Environment – Pull-Oriented)

Description The ability of a workflow case to respond to requests for data elements from a service or resource in the operating environment.

Example

- Each case of the *Handle Visa* workflow must respond to data requests from the *Immigration Department*.

Motivation The rationale for this pattern is similar to that for pattern 19 in terms of supporting data passing from a case to a resource or service external to the workflow, however in this case the data passing is initiated by a request from the external process.

Implementation For workflow engines supporting case level data, the most widely common approach to supporting this pattern is to provide the ability for external processes to interactively query case-level data elements. This pattern is not widely implemented amongst the workflow systems examined in Section 5. Only FLOWer provides direct support via the chp_dat_getval API call.

Issues None observed.

Solutions N/A

Pattern 22 (Data Interaction – Workflow to Environment – Push-Oriented)

Description The ability of a workflow engine to pass data elements to resources or services in the operational environment.

Example

- At any time the *Process Tax Return* workflow may save its working data to an external data warehouse facility.

Motivation Where a workflow engine stores data elements at workflow level, it is desirable if facilities are available for communicating this information to external applications. This is particularly useful as a means of enabling external applications to be kept informed of aggregate workflow relevant data (eg. % successful cases, resource usage etc.) without needing to continually poll the workflow engine.

The various approaches to passing workflow level data to and from the external environment are illustrated in Figure 15.

Implementation Whilst this pattern serves as a useful mechanism for proactively communicating data elements and runtime information to an external application on a whole of workflow basis, it is not widely implemented. MQSeries Workflow provides a limited ability to communicate the creation of and changes to run-time data elements (e.g. work items) to an external application via its push data access model.

Issues None observed.

Solutions N/A

Pattern 23 (Data Interaction – Environment to Workflow – Pull-Oriented)

Description The ability of a workflow to request workflow-level data elements from external applications.

Example

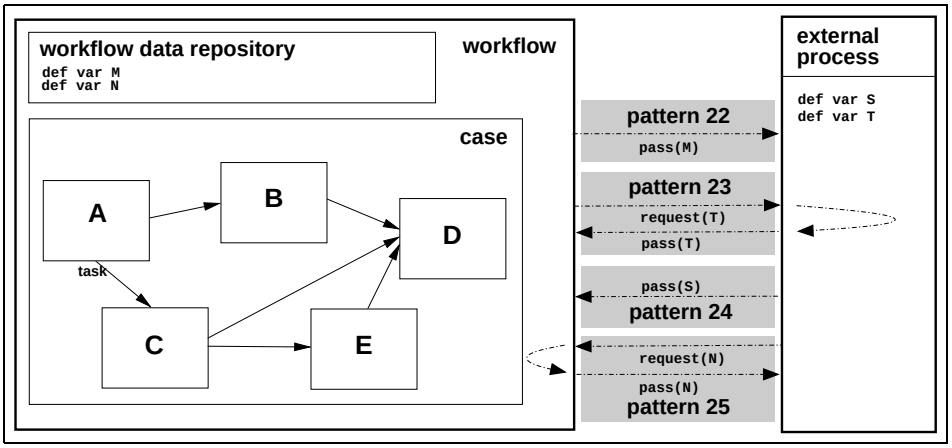

Figure 15: Data interaction between a workflow system and the operating environment

- The *Monitor Portfolios* workflow is able to request *new market position* download from the *Stock Exchange* at any time.

Motivation This pattern is motivated by the need for a workflow (or elements within it) to request data from external applications for subsequent use by some or all components of the workflow engine.

Implementation None of the workflow systems examined provide direct support for this pattern.

Issues None observed.

Solutions N/A.

Pattern 24 (Data Interaction – Environment to Workflow – Push-Oriented)

Description The ability of services or resources in the operating environment to pass workflow-level data to a workflow process.

Example

- All *mining permits* data currently available is passed to the *Develop Mining Leases* workflow.

Motivation The rationale for this pattern is to provide applications independent of the workflow engine with the ability to create or update workflow-level data elements in a workflow engine.

Implementation There are three ways in which this objective might be achieved, all of which require support from the workflow engine:

1. Data elements are passed into the workflow engine (typically as part of the command line) at the time the workflow engine is started. (This assumes that the external application starts the workflow engine.)

2. The external application initiates an import of workflow-level data elements by the workflow engine.

3. The external application utilises an API call to set data elements in the workflow engine.

Staffware provides limited support for the first of these options to set workflow configuration data. It also supports the second of these options in a limited sense through the *swutil* option which allows workflow tables and lists to be populated from external data files in predefined formats.

MQSeries Workflow provides support for the third alternative and allows peristent lists and other runtime workflow data elements to be updated via API calls.

Issues None observed.

Solutions N/A.

Pattern 25 (Data Interaction – Workflow to Environment – Pull-Oriented)

Description The ability of a workflow engine to handle requests for workflow-level data from external applications.

Example

- At any time, the *Monitor Portfolios* workflow may be required to respond to requests to provide data on any portfolios being reviewed to the *Securities Commissioner*.

Motivation Similar to the previous pattern, however in this instance, the request for workflow level data is initiated by the external application.

Implementation Generally there are two distinct approaches to achieving this requirement:

1. External applications can call utility programs provided by the workflow engine to export workflow-level data to a file which can be subsequently read by the application.

2. External applications can utilise API facilities provided by the workflow engine to access the required data programmatically.

Staffware adopts the first of these approaches to provide external applications with access to table and list data that is stored persistently by the workflow engine. MQSeries Workflow utilises the latter strategy allowing external applications to bind in API calls providing access to workflow-level data items such as persistent lists and lists of work items.

Issues None observed.

Solutions N/A.

2.4 Data Transfer Mechanisms

In this section, we consider the manner in which the actual transfer of data elements occurs between one workflow component and another. These patterns serve as an extension to those presented in Section 2.3 and aim to capture the various mechanisms by which data elements can be passed across the interface of a workflow component.

The specific style of data passing that is used in a given scenario depends on a number of factors including whether the two components share a common address

space for data elements, whether it is intended that a distinct copy of an element is passed as against a reference to it and whether the component receiving the data element can expect to have exclusive access to it. These variations give rise to a number of distinct patterns as described below.

Pattern 26 (Data Transfer by Value – Incoming)

Description The ability of a workflow component to receive incoming data elements by value relieving it from the need to have shared names or common address space with the component(s) from which it receives them.

Example

- At commencement, the *Identify Successful Applicant* task receives values for the *required role* and *salary* data elements.

Motivation Under this scenario, data elements are passed as values between communicating workflow components. There is no necessity for each workflow component to utilise a common naming strategy for the data elements or for components to share access to a common data store in which the data elements reside. This enables individual components to be written in isolation without specific knowledge of the manner in which data elements will be passed to them or the context in which they will be utilised.

Implementation This approach to data passing is commonly used for communicating data elements between tasks that do not share a common data store or wish to share task-level (or block-level) data items. The transfer of data between workflow components is typically based on the specification of mappings between them identifying source and target data locations. In this situation, there is no necessity for common naming or structure of data elements as it is only the data values that are actually transported between interacting components.

Figure16 illustrates the passing of the value of data element R in task instance A to task instance B where it is assigned to data element S. In this example, a transient variable G (depending on the specific workflow engine, this could be a data container or a case or workflow level variable) is used to mediate the transfer of the data value from task instance A to task instance B which do not share a common address space.

MQSeries Workflow utilises this approach to data passing in conjunction with distinct data channels. Data elements from the originating workflow task instance are coalesced into a data container. A mapping is defined from this data container to a distinct data container which is transported via the connecting data channel between the communicating tasks. A second mapping is then defined from the data container on the data channel to a data container in the receiving task.

BPEL4WS provides the option to pass data elements between activities using messages – an approach which relies on the transfer of data between workflow components by value.

XPDL provides more limited support for data transfer by value between a block task and subflow. As all data elements are case level, there is no explicit data passing between tasks.

Issues None observed.

Solutions N/A.

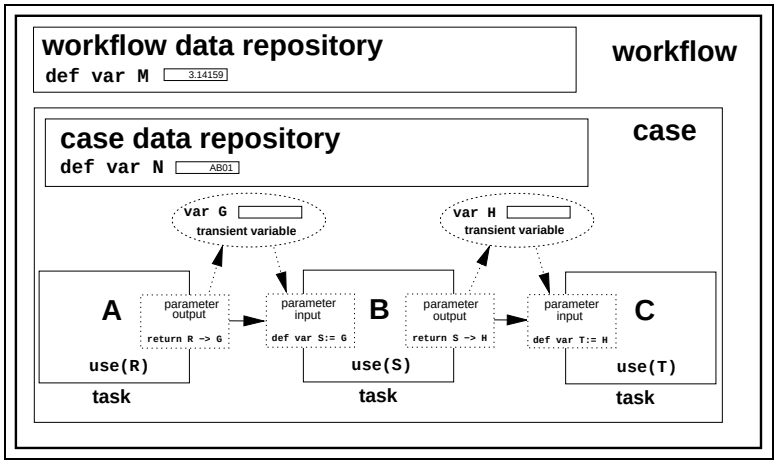

Figure 16: Data transfer by value

Pattern 27 (Data Transfer by Value – Outgoing)

Description The ability of a workflow component to pass data elements to subsequent components as values relieving it from the need to have shared names or common address space with the component(s) to which it is passing them.

Example

- Upon completion, the *Identify Successful Applicant* task passes the *successful applicant name* to the next task.

Motivation Similar to pattern 26 although in this scenario, the emphasis is on minimising the coupling between a component and the subsequent components that may receive its output data.

Implementation As for pattern 26.

Issues None observed.

Solutions N/A.

Pattern 28 (Data Transfer – Copy In/Copy Out)

Description The ability of a workflow component to copy the values of a set of data elements into its address space at the commencement of execution and to copy their final values back at completion.

Example

- When the *Review Witness Statements* task commences, copy in the *witness statement* records and copy back any changes to this data at task completion.

Motivation This facility provides components with the ability to make a local copy of data elements that can be referenced elsewhere in the workflow. This copy can then be utilised during execution and any changes that are made can be copied back at completion. It enables components to function independently of data changes and concurrency constraints that may occur in the broader workflow context.

Implementation Whilst not a widely supported data passing strategy, some workflow engines do offer limited support for it. In some cases, its use necessitates the adoption of the same naming strategy and structure for data elements within the component as used in the environment from which their values are copied. The manner in which this style of data passing operates is shown in Figure 17.

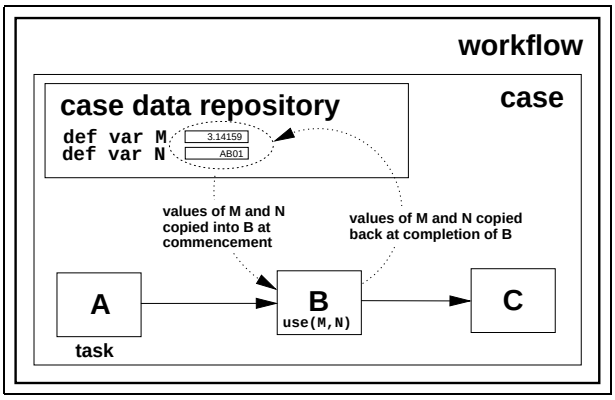

Figure 17: Data transfer – copy in/copy out

FLOWer utilises this strategy for accessing data from external databases via the InOut construct. XPDL adopts a similar strategy for data passing to and from subflows.

Issues Difficulties can arise with this data transfer strategy where data elements are passed to a sub-workflow that executes independently (asynchronously) of the calling workflow. In this situation, the calling workflow continues to execute once the sub-workflow call has been made and this can lead to problems when the sub-workflow completes and the data elements are copied back to the calling workflow as the point at which this copy back will occur is indeterminate.

Solutions There are two potential solutions to this problem:

- Do not allow asynchronous sub-workflow calls.
- In the event of asynchronous sub-workflow calls occuring, do not copy back data elements at task completion – this is the approach utilised by XPDL.

Pattern 29 (Data Transfer by Reference – Unlocked)

Description The ability to communicate data elements between workflow components by utilising a reference to the location of the data element in some mutually accessible location. No concurrency restrictions apply to the shared data element.

Example

- The *Finalise Interviewees* task passes the location of the *interviewee shortlist* to all subsequent tasks in the *Hire* process.

Motivation This pattern is commonly utilised as a means of communicating data elements between workflow components which share a common data store. It involves the use of a named data location (which is generally agreed at design time) that is

accessible to both the origin and target components and, in effect, is an implicit means of passing data as no actual transport of information occurs between the two components.

Implementation Reference-based data passing requires the ability for communicating workflow components to have access to a common data store and to utilise the same reference notation for elements that they intend to use co-jointly. There may or may not be communication of the location of shared data elements via the control channel or data channel (where supported) at the time that control flow passes from one component to the next. This mechanism for data passing is employed within the Staffware, FLOWer and COSA workflow engines and also within XPDL and BPEL4WS.

Figure 18 illustrates the passing of two data elements M and N (workflow and case level respectively) by reference from task instance A to B. Note that task instance B accepts these elements as parameters to internal data elements R and S respectively.

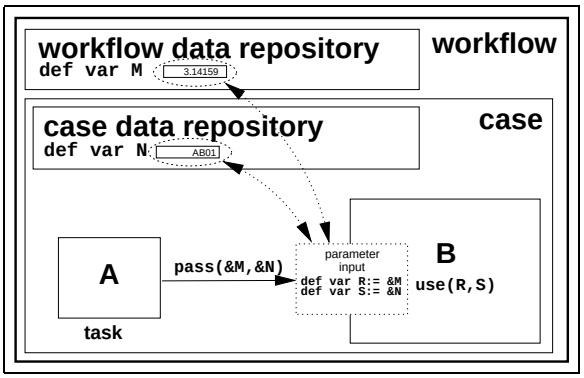

Figure 18: Data transfer by reference – unlocked

Issues None observed.

Solutions N/A.

Pattern 30 (Data Transfer by Reference – With Lock)

Example The ability to communicate data elements between workflow components by passing a reference to the location of the data element in some mutually accessible location. Concurrency restrictions are implied with the receiving component receiving the privilege of read-only or dedicated access to the data element.

Example

- At conclusion, the *Allocate Client Number* task passes the locations of the *new client number* and *new client details* data elements to the *Prepare Insurance Document* task, which receives dedicated access to these data items.

Motivation This approach is an extension of pattern 29 identified above in which there is also the expectation that the originating workflow component had acquired either read-only or exclusive access to the specific data elements being passed (i.e. a read or write lock). The workflow component that receives these data elements can choose to relinquish this access level (thus making them available to other workflow

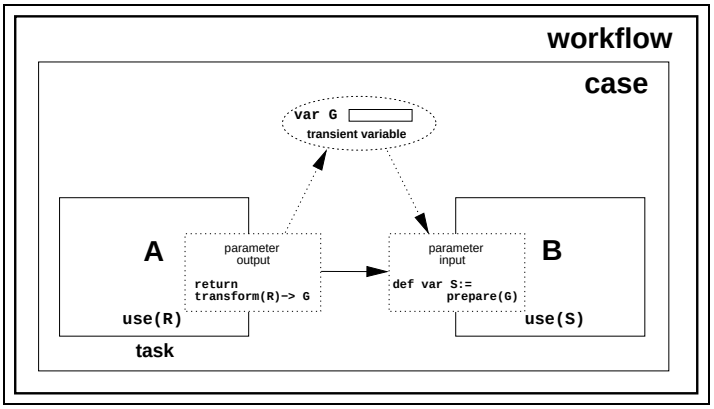

Figure 19: Data transformation – input

components) or it may choose to retain it and pass it on to later components. There is also the potential for the access level to be promoted (i.e. from a read to a write lock) or demoted (i.e. from a write to a read lock).

Implementation This pattern is not widely implemented in the workflow engines examined in Section 5. BPEL4WS provides an approach to concurrency control through the use of serializable scopes which allow fault and compensation handlers to be defined for groups of process activities. FLOWer provides limited support for a style of write lock which allows the primary case to modify data whilst still enabling it to be accessible for reading by other processes.

Issues None observed.

Solutions N/A.

Pattern 31 (Data Transformation – Input)

Description The ability to apply a transformation function to a data element prior to it being passed to a workflow component.

Example

- Prior to passing the *transform voltage* data element to the *Project_demand*() function, convert it to standard ISO measures.

Motivation The ability to specify transformation functions provides a means of handling potential mismatches between data elements and formal parameters to workflow components in a declarative manner. These functions could be internal to the workflow or could be externally facilitated.

Implementation In the example shown in Figure 19, the `prepare()` function intermediates the passing of the data element G between task instances A and B. At the point at which control is passed to B, the results of performing the `prepare()` transformation function on data element G are made available to the input parameter S.

Staffware provides the ability for a task to call a function capable of manipulating the data elements passed to the task prior to its commencement through the use of the form initial facility. The function called may be based on the Staffware script language

or external 3GL capabilities and hence can support relatively complex transformation functionality. FLOWer provides limited facilities for the transformation of input data elements through the use of mappings and derived elements.

Issues None observed.

Solutions N/A.

Pattern 32 (Data Transformation – Output)

Description The ability to apply a transformation function to a data element immediately prior to it being passed out of a workflow component.

Example

- Summarise the *spatial telemetry data* returned by the *Satellite Download* task before passing to subsequent activities.

Motivation As above.

Implementation As described for pattern 31 except that the facilities identified are used for transforming the data elements passed from a task instance rather than for those passed to it.

Issues None observed.

Solutions N/A.

2.5 Data-based Routing

Whereas previous sections have examined characteristics of data elements in isolation from other workflow perspectives (i.e. control, resource, organisational etc.), the following patterns capture the various ways in which data elements can interact with other perspectives and influence the overall operation of the workflow.

Pattern 33 (Task Precondition – Data Existence)

Description Data-based preconditions can be specified for tasks based on the presence of data elements at the time of execution.

Example

- Only execute the *Run Backup* task when *tape_loaded_flag* exists.

Motivation The ability to deal with missing data elements at the time of task invocation is desirable. This allows corrective action to be taken during workflow execution rather than necessitating the raising of an error condition and halting action.

Implementation Typically data existence preconditions are specified on task input parameters[11] in the workflow design model as illustrated in Figure 20.

In this context, data existence refers to the ability to determine whether a required parameter has been defined and provided to the task at the time of task invocation and whether it has been assigned a value.

One of five actions are possible where missing parameters are identified:

[11] For the purposes of this discussion, we use the term *parameters* in a general manner to refer both to data elements that are formally passed to a task and also to those that are shared between a task and its predecessor and passed *implicitly*.

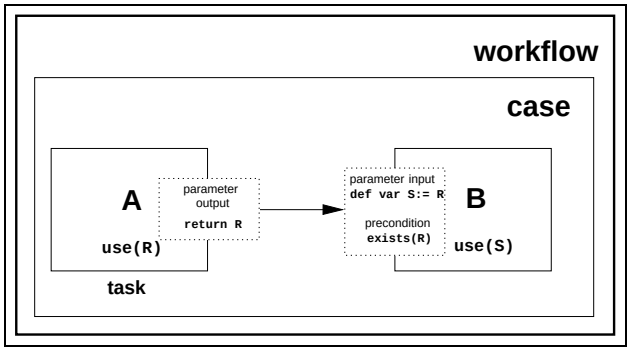

Figure 20: Task precondition – data existence

- Defer task commencement until they are available.
- Specify default values for parameters to take when they are not available.
- Request values for them interactively from workflow users.
- Skip this task and trigger the following task(s).
- Kill this thread of execution in the workflow case.

This pattern is implemented in a variety of different ways amongst several of the tools examined. Staffware provides the ability to set default values for fields which do not have a value recorded for them via the initial form command facility but only for those tasks that have forms associated with them. Conditional actions can also be used to route control flow around (i.e. skip) a task where required data elements are not available.

FLOWer provides the milestone construct which, amongst other capabilities, provides data synchronisation support allowing the commencement of a subsequent task to be deferred until nominated data elements have a value specified. COSA also provides transition conditions on incoming arcs and in conjunction with the CONDITION.TOKEN and STD tool agents, it enables the passing of control to a task to be delayed until the transition condition (which could be data element existence) is achieved.

BPEL4WS provides exception handling facilities where an attempt to utilise an uninitialised variable is detected. These can be indirectly used to provide data existence precondition support at task level but require each task to be defined as having its own scope with a dedicated fault handler to manage the uninitialised parameters accordingly.

Issues A significant consideration in managing preconditions that relate to data existence is being able to differentiate between data elements that have an undefined value and those that are merely empty or null.

Solutions Staffware addresses this issue by using distinct tokens (SW_NA) for data elements that have undefined values. Uninitialised fields have this value by default and it can be tested for in workflow expressions and conditions. FLOWer and BPEL4WS provide internal capabilities to recognise uninitialised data elements although these facilities are not directly accessible to workflow developers.

Other workflow engines examined do not differentiate between undefined and empty (or null) values.

Pattern 34 (Task Precondition – Data Value)

Description Data-based preconditions can be specified for tasks based on the value of specific parameters at the time of execution.

Example

- Execute the *Rocket Initiation* task when *countdown* is 2.

Motivation The ability to specify value-based preconditions on parameters to tasks provides the ability to delay execution of the task (possibly indefinitely) where a precondition is not satisfied.

Implementation There are three possible alternatives where a value-based precondition is not met:

- The task can be skipped and the subsequent task(s) initiated.
- Commencement of the task can be delayed until the required precondition is achieved.
- This thread of execution in the workflow case can be terminated.

This pattern is directly implemented by FLOWer through the milestone construct which enables the triggering of a task to be delayed until a parameter has a specified value. Similarly COSA provides the ability to delay execution of a task where a precondition is not met through the specification of transition conditions based on the required data values on the incoming arc to the task. By specifying alternate data conditions (which correspond to the negation of the required data values) on the outgoing arc from the state preceding the contingent task to the subsequent state, it is also possible to support the skipping of tasks where data preconditions are not met. Both approaches assume the transition conditions are specified in conjunction with the CONDITION.TOKEN tool agent.

In a more limited way, Staffware provides the ability to delay task execution through the specification of scripts which test for the required data values at task commencement. However, this approach is only possible for tasks which have forms associated with them. A better solution is to use condition actions to test for required parameters and skip the task invocation where they are not available.

XPDL provides support for a task to be skipped when a nominated data value is not achieved through the specification of an additional edge in the workflow schema which bypasses the task in question (i.e. it links the preceding and subsequent tasks) and has a transition condition which is the negation of required data values.

A similar effect can be achieved in BPEL4WS through the use of link conditions. By specifying a link condition to the task (call it A) which corresponds to the required data values and creating a parallel *empty* task in the business process that has a link condition that is the negation of this, task A will be skipped if the required data values are not detected.

Issues None identified.

Solutions N/A.

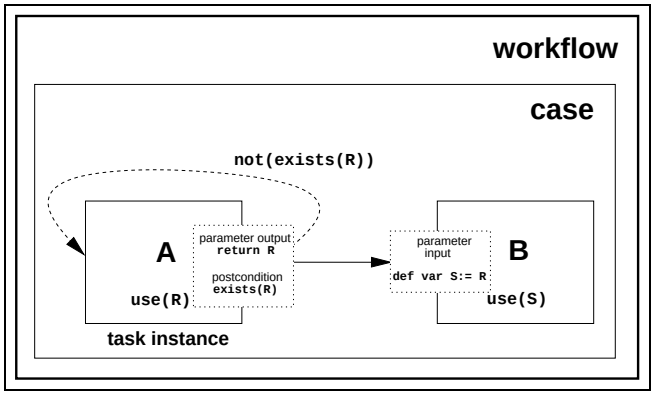

Figure 21: Task postcondition – data existence

Pattern 35 (Task Postcondition – Data Existence)

Description Data-based postconditions can be specified for tasks based on the existence of specific parameters at the time of execution.

Example

- Do not complete the *Rocket Initiation* task until *ignition data* is available.

Motivation Implementation of this pattern ensures that a task cannot complete until specified output parameters exist and have been allocated a value.

Implementation Two alternatives exist for the implementation of this pattern. Where tasks that have effectively finished all of their processing but have a nominated data existence postcondition that has not been satisfied, either the task could be suspended until the required postcondition is met or the task could be implicitly repeated until the specified postcondition is met.

Figure 21 illustrates this pattern. The specification of a data-based postcondition on a task effectively establishes an implicit control loop back to the beginning of the task that corresponds to the negation of the postcondition. Depending on the semantics of control flow within a specific workflow engine, the re-routing of control back to the beginning of the task may or may not result in the task being executed again. The implication of both scenarios however, is that the task does not pass on control flow until the required parameters exist and have defined values.

FLOWer provides direct support for this pattern by allowing data element fields in task constructs (called plan elements) to be specified as mandatory, thus ensuring they have a value specified before the plan element can complete execution. MQSeries Workflow implements this pattern through the specification of exit conditions on tasks. If the required exit condition is not met at the time task processed is completed, then the task is restarted. A specific IS NULL function is available to test if a parameter has been assigned a value.

Staffware provides indirect support through the inclusion of release scripts with tasks which evaluate whether the required data elements have been specified. Completion of the task is deferred until the required condition is satisfied although this approach is limited to tasks that have forms associated with them.

Issues As for pattern 33.

Solutions As for pattern 33.

Pattern 36 (Task Postcondition – Data Value)

Description Data-based postconditions can be specified for tasks based on the value of specific parameters at the time of execution.

Example

- Execute the *Fill Rocket* task until *rocket-volume* is 100%.

Motivation Implementation of this pattern would ensure that a task could not complete until specific output parameters had a particular data value or are in a specified range.

Implementation Similar to pattern 35, two options exist for handling the achievement of specified values for data elements at task completion:

- Delay execution until the required values are achieved.
- Implicitly re-run the task.

The implementation methods for this pattern adopted by the various tools examined are identical to those described in pattern 35.

Issues None identified.

Solutions N/A.

Pattern 37 (Event-based Task Trigger)

Description The ability for an external event to initiate a task.

Example

- Initiate the *Emergency Shutdown* task immediately after the *Power Alarm* event occurs.

Motivation This pattern centres on the ability of an external event to trigger the initiation or resumption of a specific workflow task instance. This enables the control flow of the workflow to be influenced by external applications.

Implementation This facility generally takes the form of an external workflow interface that provides a means for applications to trigger the execution of a specific task instance.

There are three distinct scenarios that may arise in the context of this pattern as illustrated in Figure 22.

The first alternative (illustrated by the *start_A*() function in Figure 22) is that the task instance to be initiated is the first task in the workflow. This is equivalent in control terms to starting a new workflow case in which A is the first task instance.

The second alternative (illustrated by the *start_B*() function) is that the external event is triggering the resumption of a task instance that is in the middle of a workflow process. The task instance has already had control passed to it but its execution is suspended pending occurrence of the external event trigger. This situation is shown in Figure 22 above with task instance B already triggered as a result of task instance A completing but halted from further progress until the event from *start_B*() occurs.

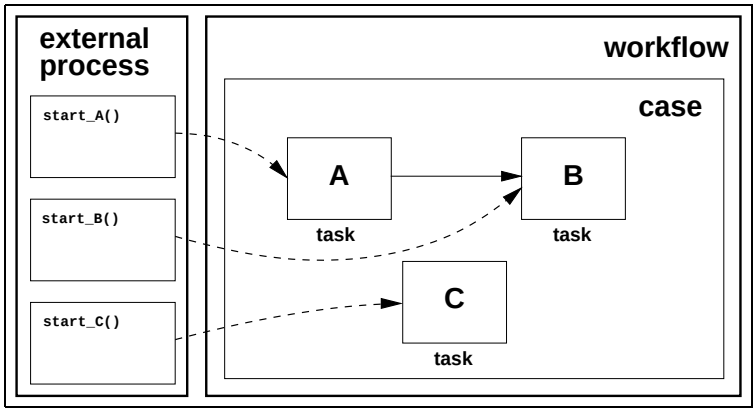

Figure 22: Event-based task trigger

The third alternative is that the task instance is isolated from the main control flow in the workflow and the only way in which it can be initiated is by receiving an external event stimulus. Figure 22 shows task instance C which can only be triggered when the event stimulus from $start\_C()$ is received.

All three variants of this pattern are directly supported by Staffware, FLOWer, COSA and BPEL4WS and in all cases, the passing of data elements as well as process triggering is supported. MQSeries Workflow provides indirect support for all three variants but requires event handling to be explicitly coded into activity implementations.

Issues None identified.

Solutions N/A.

Pattern 38 (Data-based Task Trigger)

Description The ability to trigger a specific task when an expression based on workflow data elements evaluates to true.

Example

- Trigger the *Re-balance Portfolio* task when the *loan margin is less than 85%.*

Motivation This pattern provides a means of triggering the initiation or resumption of a task instance when a condition based on workflow data elements is satisfied.

Implementation This pattern is analogous to the notion of active rules [Har93] or event-condition-action (ECA) rules [PD99] found in active databases [DGG95].

This pattern is directly supported in FLOWer through the specification of a condition corresponding to the required data expression on a milestone construct immediately preceding the to-be-triggered task. When the data condition is met, the task is then triggered.

Similarly the pattern can be directly implemented in COSA through the use of transition conditions which incorporate the data condition being monitored for on the incoming edges to the to-be-triggered task. Depending on the semantics required

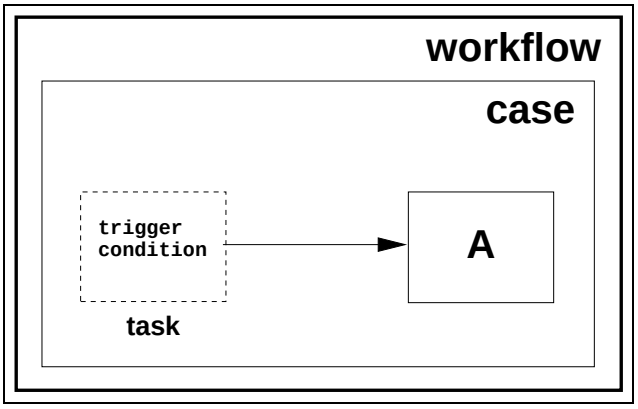

Figure 23: Data-based task trigger

for the triggering, the transition condition may or may not include the CONDITION.TOKEN tool agent[12].

Many workflow systems do not directly support this pattern, however in some cases it can be constructed for workflows that support event-based triggering (i.e. pattern 37) by simulating event-based triggering within the context of the workflow. This is achieved by nominating an event that the triggered task should be initiated/resumed on and then establishing a data monitoring task that runs in parallel with all other workflow tasks and monitors data values for the occurrence of the required triggers. When one of them is detected, the task that requires triggering is initiated by raising the event that it is waiting on.

The only caveat to this approach is that the workflow must not support the *implicit termination* pattern [AHKB03]. If the workflow were to support this pattern, then problems would arise for each workflow case when the final task was completed as this would not imply that other outstanding task instances should also terminate. Since the monitoring task could potentially run indefinitely, it could not be guaranteed that it would cease execution at case completion.

This scenario is illustrated in Figure 23. Task instance A is to be triggered when *trigger condition* evaluates to true. A task instance is set up to monitor the status of *trigger condition* and to complete and pass control to A when it occurs.

By adopting this strategy, the pattern can be indirectly implemented in BPEL4WS. Although it could be similarly constructed in Staffware and variants which replaced the event link with a control edge could be implemented in MQSeries Workflow and XPDL, all of these workflows support implicit termination and hence would lead to problematic implementations of this pattern.

Issues None identified.

Solutions N/A.

Pattern 39 (Data-based Routing)

Description The ability to alter the control flow within a workflow case as a consequence of the value of data-based expressions.

[12]If it does include this condition, the to-be-triggered task will only run when there is a token in the immediately preceding state and the transition condition is met. If not, the task will become executable whenever the transition condition is met.

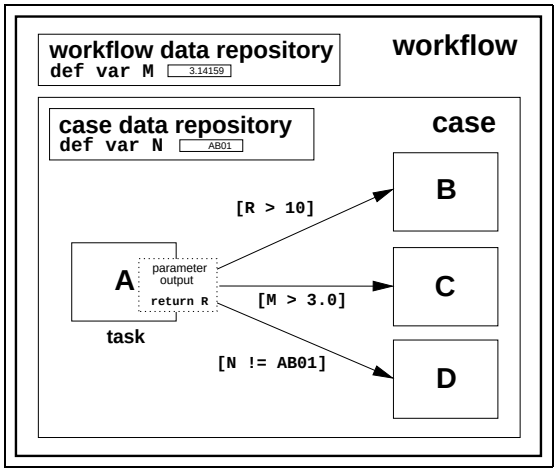

Figure 24: Data-based routing

Example

- If *alert* is *red* then execute the *Inform Fire Crew* task after the *Handle Alert* task otherwise run the *Identify False Trigger* task.

Motivation Without data-based routing constructs, it would not be possible for a workflow to alter its operation in response to the data it was processing. In effect, the function of the workflow would be rigidly fixed at design time. Data-based routing provides the ability for the completion of one task instance to trigger one or several subsequent task instances depending on the outcome of an expression based on the values of various workflow data elements. Figure 24 illustrates the routing expressions as constructs on the control channel from one task instance to another. Data in these expressions can be task-level data passed from the completing task instance, case or workflow level data elements. In the example shown, task instance C will be triggered once A completes (as the value of the workflow level data element M is greater than 3.0), task instance D will not and task instance B may be triggered depending on the value of data element R that is passed from A.

Implementation This pattern serves as an aggregation of two major control-flow patterns [AHKB03] that depend on workflow data:

- *exclusive choice* – where control flow is passed to one of several subsequent task instances depending on the outcome of a decision or the value of an expression based on workflow data elements.

- *multi-choice* – where, depending on the outcome of a decision or the value of an expression based on workflow data elements, control flow is passed to several subsequent task instances.

Both variants of this construct are supported by MQSeries Workflow, COSA, XPDL and BPEL4WS. Staffware and FLOWer only support the exclusive choice pattern.

Issues None identified.

Solutions N/A.

3 Survey of existing workflow systems and business process languages

This section presents the results of a detailed evaluation of support for the 39 workflow data patterns described above by six workflow systems and business process languages. A broad range of offerings were chosen for this review in order to validate the applicability of each of the patterns to the various types of tools that fall under the "workflow umbrella" [GHS95].

In summary, the tools evaluated were:

- **Staffware Process Suite version 9**[13] [Sta02a, Sta02b] – a widely used fully-integrated commercial workflow system.
- **MQSeries Workflow 3.3.2**[14] [IBM01a, IBM01b] – another widely used commercial workflow system that coordinates tasks based on 3GL programs.
- **FLOWer 2.05** [Wf02] – a case handling system that adopts a data-driven approach to workflow execution.
- **COSA 4.2** [TRA03, TRA03] – a workflow system based on the Petri-Net modelling formalism.
- **XPDL 1.0** [Wor02] – a process definition language for workflow proposed by the WfMC.
- **BPEL4WS 1.1** [ACD+03] – a language for specifying business process execution based on web services.

A three scale assessment scale is used with + indicating direct support for the pattern, +/– indicating partial support and – indicating that the pattern is not implemented. The specifics of the rating criteria used are contained in appendix G.

Table 2 lists the first set of results which relate to the various levels of data construct visibility supported within the tool. As a general rule, it can be seen that individual products tend to favour either a task-level approach to managing production data and pass data elements between task instances or they use a shared data store at case level. The only exception to this being COSA which fully supports data at both levels.

A similar result can be observed for workflow and environment data with most workflow products fully supporting one or the other (Staffware being the exception here). The implication of this generally being that globally accessible data can either be stored in the workflow product or outside of it (i.e. in a database). XPDL and BPEL4WS are the exceptions although this outcome seems to relate more to the fact that there is minimal consideration for global data facilities within these specifications.

[13] Although not the latest version, the functionality from a data perspective is representative of the latest offering.

[14] Recently renamed Websphere MQ Workflow. The product is essentially unchanged from the version examined.

Nr	Pattern	*Staffware*	*MQSeries*	*FLOWer*	*COSA*	*XPDL*	*BPEL4WS*
1	Task Data	–	+/–	+/–	+	–	+/–
2	Block Data	+	+	+	+	+	–
3	Scope Data	–	–	+/–	–	–	+
4	Multiple Instance Data	+/–	+	+	+	+	–
5	Case Data	+/–	+	+	+	+	+
6	Workflow Data	+	+	–	+/–	+/–	–
7	Environment Data	+	+/–	+	+	–	+

Table 2: Support for Data Visibility Patterns in Workflow Systems

Table 3 lists the results for internal data passing. All products supported task-to-task interaction and block task-to-sub-workflow interaction[15]. The notable omissions here were the general lack of support for handling data passing to multiple instance tasks (FLOWer being the exception) and the lack of integrated support for data passing between cases.

Nr	Pattern	*Staffware*	*MQSeries*	*FLOWer*	*COSA*	*XPDL*	*BPEL4WS*
8	Data Interaction between Tasks	+	+	+	+	+	+
9	Data Interaction – Block Task to Sub-workflow Decomposition	+	+	+/–	+/–	+	–
10	Data Interaction – Sub-workflow Decomposition to Block Task	+	+	+/–	+/–	+	–
11	Data Interaction – to Multiple Instance Task	–	–	+	–	–	–
12	Data Interaction – from Multiple Instance Task	–	–	+	–	–	–
13	Data Interaction – Case to Case	+/–	+/–	+/–	+/–	+/–	+/–

Table 3: Support for Internal Data Interaction Patterns in Workflow Systems

The results in Table 4 indicate the ability of the workflow products to integrate with data sources and applications in the operating environment. MQSeries Workflow, FLOWer and COSA demonstrate a broad range of capabilities in this area. XPDL and BPEL4WS clearly have limited potential for achieving external integration other than with web services.

[15]BPEL4WS being the exception given its lack of support for sub-workflows.

Nr	Pattern	*Staffware*	*MQSeries*	*FLOWer*	*COSA*	*XPDL*	*BPEL4WS*
14	Data Interaction – Task to Environment – Push-Oriented	+	+/–	+	+	+	+
15	Data Interaction – Environment to Task – Pull-Oriented	+	+/–	+	+	+	+
16	Data Interaction – Environment to Task – Push-Oriented	+/–	+/–	+/–	+	–	+/–
17	Data Interaction – Task to Environment – Pull-Oriented	+/–	+/–	+/–	+/–	–	+/–
18	Data Interaction – Case to Environment – Push-Oriented	–	–	+	–	–	–
19	Data Interaction – Environment to Case – Pull-Oriented	–	–	+	–	–	–
20	Data Interaction – Environment to Case – Push-Oriented	+/–	+/–	+	+	–	–
21	Data Interaction – Case to Environment – Pull-Oriented	–	–	+	+	–	–
22	Data Interaction – Workflow to Environment – Push-Oriented	–	+/–	–	–	–	–
23	Data Interaction – Environment to Workflow – Pull-Oriented	+/–	–	–	–	–	–
24	Data Interaction – Environment to Workflow – Push-Oriented	–	+/–	–	–	–	–
25	Data Interaction – Workflow to Environment – Pull-Oriented	+	+	–	–	–	–

Table 4: Support for External Data Interaction Patterns in Workflow Systems

Table 5 illustrates the mechanisms used by individual workflow engines for passing data between components. Generally this occurs by value or by reference. There are two areas where there is clear opportunity for improvement. First, support for concurrency management where data is being passed between components – only FLOWer and BPEL4WS offered some form of solution to this problem. Second, the transformation of data elements being passed between components – only Staffware provides a fully functional capability for dealing with potential data mismatches between sending and receiving components although its applicability it limited.

Table 6 indicates the ability of the data perspective to influence the control perspective within each product. FLOWer demonstrates outstanding capability in this area and Staffware, MQSeries Workflow and COSA also have relatively good integration of the data perspective with control flow although each of them lack some degree of task pre and postcondition support. Similar comments apply to XPDL which has significantly more modest capabilities in this area and completely lacks any form of

Nr	Pattern	*Staffware*	*MQSeries*	*FLOWer*	*COSA*	*XPDL*	*BPEL4WS*
26	Data Passing by Value – Incoming	–	+	–	–	+/–	+
27	Data Passing by Value – Outgoing	–	+	–	–	+/–	+
28	Data Passing – Copy In/Copy Out	–	–	+/–	–	+/–	–
29	Data Passing by Reference – Unlocked	+	–	+	+	+	+
30	Data Passing by Reference – Locked	–	–	+/–	–	–	+/–
31	Data Transformation – Input	+/–	–	+/–	–	–	–
32	Data Transformation – Output	+/–	–	+/–	–	–	–

Table 5: Support for Data Transfer Patterns in Workflow Systems

Nr	Pattern	*Staffware*	*MQSeries*	*FLOWer*	*COSA*	*XPDL*	*BPEL4WS*
33	Task Precondition – Data Existence	+	–	+	+	–	+/–
34	Task Precondition – Data Value	+	–	+	+	+	+
35	Task Postcondition – Data Existence	+/–	+	+	–	–	–
36	Task Postcondition – Data Value	+/–	+	+	–	–	–
37	Event-based Task Trigger	+	+/–	+	+	–	+
38	Data-based Task Trigger	–	–	+	+	–	+/–
39	Data-based Routing	+/–	+	+/–	+	+	+

Table 6: Support for Data Routing Patterns in Workflow Systems

trigger support. BPEL4WS would also benefit from better pre and postcondition support and lacks data-based triggering.

4 Discussion

The rationale for this work was to undertake a comprehensive review of the various ways in which data was utilised in workflow systems and to catalogue the results of this investigation. In order to ensure the findings were not specific to any particular modelling formalism or implementation technology, a patterns-based approach was taken to documenting the various data concepts that were identified. This strategy had the dual benefits of both ensuring that the data patterns identified were independent of any specific language and also continuing the definition of workflow patterns for the data perspective at the same conceptual level as previous work undertaken for the control perspective (see [AHKB03] for details).

One of the main aims in developing patterns for this workflow perspective was to establish a suitable basis for comparison between distinct workflow implementation technologies of the data support facilities that they provide. To this end, it was important that the patterns were generalisable across the broad range of workflow offerings and that they were not reliant on any particular conceptual underpining. In order to validate the applicability of these patterns, we utilised them as a basis for the evaluation and comparison of three commercial workflow offerings (Staffware, MQSeries Workflow and COSA) and to establish their applicability to activity-centric tools more generally, we extended this review to also include a case handling system (FLOWer), a process definition language (XPDL) and a business process execution language (BPEL4WS). The results of these evaluations have been encouraging and clearly show that the patterns identified have general applicability across the spectrum of activity centric tools.

Criticism of the patterns approach often centres on its prescriptive nature and the implications of demonstrating or not demonstrating compliance with specific patterns at tool level. Our view is that these concerns are misguided. The aim of this work is to establish a broad catalogue of the data concepts that are relevant to workflow systems and to activity-centric systems more generally. This has been undertaken through a systematic evaluation of currently available commercial products. Whilst it has led to the definition and validation of 39 data patterns, there is no claim that this represents the complete set of data relevant concepts for workflow systems or that all patterns are equally important. What is clear is that these patterns do provide a basis for assessing the *capabilities* of individual offerings from a data perspective.

Demonstration of compliance with all data patterns by an individual offering should not be viewed as the pre-eminent objective for vendors. Indeed it is clear even from the most cursory glance of Section 5 that no tool achieves this. Of more interest is the issue of *suitability* i.e. does demonstration of a specific group of patterns indicate that a tool is suitable for a specific purpose e.g. enterprise application integration, web services orchestration or case handling. In order to better understand this problem, it is necessary to establish clear criteria for individual functional areas from a patterns perspective e.g. what is the minimal set of data patterns that will enable an offering to provide EAI facilities capable of integrating a variety of disparate corporate systems. By determining these correspondences, it will be possible to give better guidance as to the capabilities of individual tools. It will also simplify the selection process when a tool is sought for a specific purpose.

Future work will focus on better understanding the suitability problem as well as further identification of patterns in other workflow perspectives.

Several observations are worthy of note from the results in Section 3 in regard to the capabilities and shortcomings of existing offerings. One of the most immediate is that the level of direct support for data patterns in standard workflow design tools is minimal. These tools focus largely on the definition of process structures and their associated components in a graphical form. Support for the data perspective is typically fragmented and distributed across various parts of the overall workflow model. This situation is partially explained by the graphical nature of control flow and its primacy as a means of workflow modelling. Although there are graphical means of characterising the information contained in the data perspective (e.g. ER diagrams, class diagrams, object-role models), these are not widely utilised in current design tools and there is clearly a need for better support in capturing the integration

between the control flow and data perspectives at design time.

Another observation is that many systems/standards do not offer support for multiple instance tasks. These constructs serve as an effective model of concurrency where the underlying activities are largely based on a common set of actions. From a data perspective, this pattern is quite demanding and there are a number of considerations associated with its implementation although for many of the offerings examined it would constitute a useful addition to current modelling and implementation constructs.

A number of the offerings examined implement a shared repository for data elements (at block, scope or case level) where tasks access a common set of data elements rather than passing them between tasks. Whilst this approach has advantages from a performance standpoint and improves the overall tractability of process models, it is surprising how few of them offer an effective means of concurrency control for shared data. Indeed, the maturity of the data models supported by all of the offerings examined is well short of the state of the art in database technology and there is an evident lack of support for complex data structures, relationships between workflow cases and their associated data elements and streamlined facilities for interacting with the external environment in the various plethora of ways that the contemporary corporate application environment requires.

5 Related work

Given the prominence that patterns now enjoy within the Information Technology community, it is ironic that they were initially proposed by an Architect as a means of characterising recurring design problems and describing solutions to them that may have general applicability. In his seminal work *A Pattern Language* [AIS77], Christopher Alexander laid down the initial concept of a pattern and presented the blueprint for documenting a pattern, a format which has significantly influenced more recent patterns initiatives. Subsequently in [Ale79], he further developed the use of patterns as a design language and offered the widely cited definition of a pattern as "a quality without a name".

The *Design Patterns* [GHJV95] initiative was the first application of the patterns concept in the IT field. It catalogued 23 "important and recurring designs in object-oriented systems" and described the problems that each of them addresses, the solution in detail and any considerations that are associated with their usage. A template format was adopted in the documentation of each pattern that was particularly relevant to the IT domain and this style of presentation has significantly influenced the format and scope of later pattern initiatives.

The publication of this work and the inherent appeal of the patterns approach acted as a trigger to a number of IT practitioners to capture conceptual designs (both fragments and in complete form) that were capable of being reused. Associated with this objective was the expectation that individual patterns were "tried and tested" in at least one production system, thus ensuring that catalogues of patterns could serve as a potential source book for designers during the development process – in a similar manner to which pre-existing designs are re-used in other disciplines such as architecture and engineering.

There have been a number of significant works produced which capture various

forms of recurrent design considerations. In [BMR+96], a broad view is taken of patterns in the context of software architecture and a series of constructs are documented that "range from high-level architectural patterns through design patterns to low-level idioms". This work is extended in [SSRB00], with a further 17 patterns which are specifically focussed on concurrent and network systems.

In [Hay95], relatively complete and concrete conceptual models for applications in various domains are presented in the form of entity/relationship diagrams. Similarly [Fow96], describes a series of analysis patterns which are defined as "groups of constructs that represent a common construction in business modeling" and additional patterns which support them. Once again, these patterns are relatively concrete in format and framed with respect to specific application domains.

Other noteworthy patterns collections include [Fow03] which presents a set of patterns relevant to enterprise application design, [HW04] which centres on patterns for the design and deployment of messaging systems and [Mon03] which describes patterns for the web services domain.

6 Conclusion

This paper has identified 39 new *workflow data patterns* which describe the manner in which data elements are defined and utilised in workflow systems. Validation of the applicability of these patterns has been achieved through a detailed review of six workflow systems and business process languages.

These patterns are intended as an extension to previous work on workflow control flow patterns. The focus of the *Workflow Patterns* initiative is on identifying generic structures in various perspectives across the class of workflow systems. In doing so, it is anticipated that the key elements of a workflow description language can be identified that are applicable across the workflow domain.

Validation of these constructs also gives a valuable insight into the operation of workflow systems and offers the basis for identifying potential enhancements to individual offerings. It allows for comparison of the feature set supported by individual products to occur in a manner that is independent of the conceptual underpinnings of specific tools and gives an indicative measure of the capabilities of specific offerings.

Acknowledgments

This work was partially supported by the Australian Research Council under the Discovery Grant "*Expressiveness Comparison and Interchange Facilitation between Business Process Execution Languages*".

Disclaimer

We, the authors and the associated institutions, assume no legal liability or responsibility for the accuracy and completeness of any product-specific information contained in this paper. All possible efforts have been make to ensure that the results presented are, to the best of our knowledge, up to date and correct.

References

[AAA+96] G. Alonso, D. Agrawal, A. El Abbadi, M. Kamath, G. Gunthor, and C. Mohan. Advanced Transaction Models in Workflow Contexts. In *12th International Conference on Data Engineering*, New Orleans, Louisiana, USA, 1996.

[Aal96] W. van der Aalst. Petri-Net-Based Workflow Management Software. In *NSF Workshop on Workflow and Process Automation in Information Systems, 1996*, Athens, Georgia, USA, 1996.

[Aal98] W. van der Aalst. The Application of Petri Nets to Workflow Management. *Journal of Circuits, Systems and Computers*, 8(1):21–66, 1998.

[ACD+03] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weerawarana. Business Process Execution Language for Web Services Version 1.1. Technical report, 2003. Accessed at `http://xml.coverpages.org/BPELv11-May052003Final.pdf` on 19 April 2004.

[AH04] W.M.P. van der Aalst and A.H.M. ter Hofstede. YAWL: Yet Another Workflow Language. Accepted for publication in Information Systems, 2004.

[AHKB03] W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, and A.P. Barros. Workflow Patterns. *Distributed and Parallel Databases*, 14(3):5–51, 2003.

[AIS77] C. Alexander, S. Ishikawa, and M. Silverstein. *A Pattern Language: Towns, Buildings, Construction*. Oxford University Press, New York, 1977.

[Ale79] C. Alexander. *The Timeless Way of Building*. Oxford University Press, New York, 1979.

[BMR+96] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal. *Pattern Oriented-Software Architecture: A System of Patterns Volume 1*. Wiley, Chichester, 1996.

[DGG95] K.R. Dittrich, S. Gatziu, and A. Geppert. The Active Database Management System Manifesto: A Rulebase of ADBMS Features. In T. Sellis, editor, *2nd International Workshop on Rules in Database Systems*, volume 985 of *Lecture Notes in Computer*, pages 3–20, Athens, Greece, 1995. Springer.

[DH01] M. Dumas and A. ter Hofstede. UML Activity Diagrams as a Workflow Specification Language. In M. Gogolla and C. Kobryn, editors, *Fourth International Conference on the Unified Modeling Language (UML 2001)*, pages 76–90, Toronto, Canada, 2001.

[DHL01] U. Dayal, M. Hsu, and R. Ladin. Business Process Coordination: State of the Art, Trends, and Open Issues. In P.M.G. Apers, P. Atzeni, S. Ceri,

S. Paraboschi, K. Ramamohanarao, and R.T. Snodgrass, editors, *27th International Conference on Very Large Data Bases (VLDB 2001)*, pages 3–13, Roma, Italy, 2001. Morgan Kaufmann.

[Fow96] M. Fowler. *Analysis Patterns : Reusable Object Models*. Addison-Wesley, Boston, 1996.

[Fow03] M. Fowler. *Patterns of Enterprise Application Architecture*. Addison-Wesley, Boston, 2003.

[GHJV95] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Boston, 1995.

[GHS95] D. Georgakopoulos, M.F. Hornick, and A.P. Sheth. An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure. *Distributed and Parallel Databases*, 3(2):119–153, 1995.

[GP03] M. Georgeff and J. Pyke. Dynamic Process Orchestration. Technical report, Staffware White Paper, 2003. Accessed at `http://tmitwww.tm.tue.nl/bpm2003/download/WP%20Dynamic%20Process%20Orch%estration%20v1.pdf` on 6 April 2004.

[Gre02] P. Grefen. A Taxonomy for Transactional Workflows. Technical report, CTIT Technical Report 02-11, University of Twente, 2002.

[GVA01] P. Grefen, J. Vonk, and P. Apers. Global Transaction Support for Workflow Management Systems: From Formal Specification to Practical Implementation. *The VLDB Journal*, 10:316–333, 2001.

[Har93] J.V. Harrison. Active Rules in Deductive Databases. In *Second International Conference on Information and Knowledge Management*, pages 174–183, Washington D.C., USA, 1993. ACM Press.

[Hay95] D.C. Hay. *Data Model Patterns, Conventions of Thought*. Dorset House, New York, 1995.

[Hol95] D. Hollingsworth. The Workflow Reference Model - Issue 1.1. Technical Report Document Number TC00-1003, Workflow Management Coalition, 1995. Accessed from `http://www.wfmc.org/standards/docs/tc003v11.pdf` on 6 April 2004.

[HW04] G. Hohpe and B. Woolf. *Enterprise Integration Patterns : Designing, Building, and Deploying Messaging Solutions*. Addison-Wesley, Boston, 2004.

[IBM01a] IBM. *IBM MQSeries Workflow – Getting Started with Buildtime – Version 3.3*. IBM Corp., 2001.

[IBM01b] IBM. *IBM MQSeries Workflow – Programming Guide – Version 3.3*. IBM Corp., 2001.

[JB96] S. Jablonski and C. Bussler. *Workflow Management: Modeling Concepts, Architecture and Implementation*. Thomson Computer Press, 1996.

[KS95] N. Krishnakumar and A. Sheth. Managing Heterogeneous Multi-system Tasks to Support Enterprise-wide Operations. *Distributed and Parallel Database Systems*, 3(2):155–186, 1995.

[Mon03] P.B. Monday. *Web Services Patterns*. Apress, New York, 2003.

[OMG00] OMG. Workflow Management Facility Specification V1.2. Technical report, 2000. Accessed at `http://www.omg.org/technology/documents/formal/workflow_management.htm` on 6 April 2004.

[OMG03] OMG. OMG Unified Modeling Language Specification Revision 1.5. Technical report, OMG, 2003. Accessed at `http://www.omg.org/technology/documents/formal/uml.htm` on 25 April 2004.

[PD99] N.W. Paton and O. Diaz. Active Database Systems. *ACM Computing Surveys*, 1(31):63–103, 1999.

[RS95] M. Rusinkiewicz and A. Sheth. Specification and Execution of Transactional Workflows. In W. Kim, editor, *Modern Database Systems — The Object Model, Interoperability, and Beyond*, pages 592–620. Addison-Wesley, 1995.

[Sch00] A..-W. Scheer. *ARIS - Business Process Modelling*. Springer, Berlin, 2000.

[SSRB00] D. Schmidt, M. Stal, H. Rohnert, and F. Buschmann. *Pattern Oriented-Software Architecture: Patterns for Concurrent and Networked Objects Volume 2*. Wiley, Chichester, 2000.

[Sta02a] Staffware. *Staffware Process Suite – Defining Staffware Procedures – Issue 2*. Staffware plc, Maidenhead, 2002.

[Sta02b] Staffware. *Staffware Process Suite – Integrating Staffware with Your Enterprise Applications – Issue 2*. Staffware plc, Maidenhead, 2002.

[TRA03] TRANSFLOW. *COSA 4 Business-Process Designer's Guide*. TRANSFLOW AG, Pullheim, 2003.

[VDGK00] J. Vonk, W. Derks, P. Grefen, and M. Koetsier. Cross-Organizational Transaction Support for Virtual Enterprises. In O. Etzion and P. Scheuermann, editors, *5th International Conference on Cooperative Information Systems (CoopIS'2000)*, pages 323–334, Eilat, Israel, 2000.

[Wf02] Wave-front. *FLOWer 2.1 Designers Guide*. Wave-front BV, 2002.

[Wor02] Workflow Management Coalition. Workflow Process Definition Interface – XML Process Definition Language. Technical Report Document Number WFMC-TC-1025, Workflow Management Coalition, 2002. Accessed at `http://www.wfmc.org/standards/docs.htm` on 6 April 2004.

[Wor04] Workflow Management Coalition. Workflow Management Coalition (WfMC) Web Site, 2004. Accessed at `http://www.wfmc.org` on 6 April 2004.

[WR92] H. Wachter and A. Reuter. The ConTract Model. In A.K. Elmargarmid, editor, *Database Transaction Models for Advanced Applications*, pages 219–263. Morgan Kauffman, San Mateo, 1992.

[WS97] D. Worah and A.P. Sheth. Transactions in Transactional Workflows. In S. Jajodia and L. Kerschberg, editors, *Advanced Transaction Models and Architectures*, pages 3–34. Kluwer Academic Publishers, 1997.

A Staffware

This evaluation is based on version 9 of the Staffware Process Suite.

Nr	Pattern	Score	Motivation
1	t-d	–	Not supported.
2	b-d	+	Directly supported – each (sub)procedure can maintain a distinct set of data fields.
3	s-d	–	Not supported.
4	mi-d	+/–	Only scenario supported is multiple task triggering but there is no direct means of ensuring data independence of each invocation.
5	c-d	+/–	Main workflow procedure can maintain data fields for each case but they must be explicitly passed to sub-workflows.
6	wf-d	+	Supported through tables and lists.
7	e-d	+	Supported through script functions and Automatic Steps.
8	di-t	+	Data interaction between tasks is based on use of common block-level data fields.
9	di-b-s	+	Indirectly supported via field mappings.
10	di-s-b	+	Indirectly supported via field mappings.
11	di-a-mi	–	Multiple instance tasks not supported.
12	di-mi-a	–	Multiple instance tasks not supported.
13	di-c-c	+/–	Achievable via script functions.
14	di-t-e-push	+	Directly supported via Automatic Steps and script functions.
15	di-e-t-pull	+	Directly supported via Automatic Steps and script functions.
16	di-e-t-push	+/–	Indirectly supported by Event Steps.
17	di-t-e-pull	+/–	Indirectly supported by EIS Reports and the EIS Case Data extraction facility.
18	di-c-e-push	–	Not supported.
19	di-e-c-pull	–	Not supported.
20	di-e-c-push	+/–	Limited supported for passing parameters at case commencement.
21	di-c-e-pull	–	Not supported.
22	di-w-e-push	–	Not supported.
23	di-e-w-pull	+/–	Indirect support for importing tables and lists via the swutil function.
24	di-e-w-push	–	Not supported.
25	di-w-e-pull	+	Access to table and field data is supported via the swutil function.
26	dt-val-in	–	Not supported.
27	dt-val-out	–	Not supported.
28	dt-cico	–	Not supported.

Nr	Pattern	Score	Motivation
29	dt-r-unl	+	Achievable via fields.
30	dt-r-l	–	Not supported.
31	dt-in	+/–	Supported via form initial function for tasks that have forms.
32	dt-out	+/–	Supported via form release function for tasks that have forms.
33	t-pre-de	+	Uninitialised fields can be detected via the SW_NA value. Initial scripts can be used to set default values for missing parameters where tasks have forms. Conditional actions can be used to skip tasks where required parameters are missing.
34	t-pre-dv	+	Initial scripts can be used to delay invocation depending on parameter values where tasks have forms. Conditional actions can be used to skip tasks where parameter values are not equal to specified values.
35	t-post-de	+/–	Indirectly achievable via validation functions linked to release form commands.
36	t-post-dv	+/–	Indirectly achievable via validation functions linked to release form commands.
37	ett	+	Events are directly supported. New workflow cases can also be externally invoked.
38	dtt	–	Not supported.
39	dbr	+/–	Only exclusive choice supported.

B MQSeries Workflow

This evaluation is based on MQSeries Workflow 3.3.2.

Nr	Pattern	Score	Motivation
1	t-d	+/–	Indirectly supported via 3GL program implementations.
2	b-d	+	Supported via global data containers for a process.
3	s-d	–	Not supported.
4	mi-d	+	Support for distinct data elements in multiply triggered tasks and tasks with shared sub-workflow decompositions.
5	c-d	+	Supported through the default data structure.
6	wf-d	+	Supported through persistent lists.
7	e-d	+/–	Indirectly accessible via program implementations.
8	di-t	+	Facilitated via data containers passed between tasks on data channels.
9	di-b-s	+	Facilitated via data containers and source/sink nodes.
10	di-s-b	+	Facilitated via data containers and source/sink nodes.
11	di-a-mi	–	No support for multiple instances.
12	di-mi-a	–	No support for multiple instances.
13	di-c-c	+/–	Indirectly achievable by including case communication facilities in program implementations.
14	di-t-e-push	+/–	Indirectly achievable by including communication facilities in program implementations.
15	di-e-t-pull	+/–	Indirectly achievable by including communication facilities in program implementations.
16	di-e-t-push	+/–	Indirectly achievable by including communication facilities in program implementations.
17	di-t-e-pull	+/–	Indirectly achievable by including communication facilities in program implementations.
18	di-c-e-push	–	Not supported.
19	di-e-c-pull	–	Not supported.
20	di-e-c-push	+/–	Parameter values can be specified at the initiation of each case.
21	di-c-e-pull	–	Not supported.
22	di-w-e-push	+/–	Push data access model provides limited facilities for communicating runtime workflow data to external applications.
23	di-e-w-pull	–	Not supported.
24	di-e-w-push	+/–	Indirect support for manipulating persistent lists via API calls.
25	di-w-e-pull	+	Directly supported via the Runtime API.
26	dt-val-in	+	All data passing via containers is by value.
27	dt-val-out	+	All data passing via containers is by value.
28	dt-cico	–	Not supported.

Nr	Pattern	Score	Motivation
29	dt-r-unl	–	Not supported.
30	dt-r-l	–	Not supported.
31	dt-in	–	Not supported.
32	dt-out	–	Not supported.
33	t-pre-de	–	Not supported.
34	t-pre-dv	–	Not supported.
35	t-post-de	+	Directly supported via exit conditions and the IS NULL function which allow parameter value assignment to be tested. Where exit condition is not met, task automatically repeats.
36	t-post-dv	+	Directly supported by specifying exit condition based on the required parameter value. Where exit condition is not met, task automatically repeats.
37	ett	+/–	Activity triggering is supported through various interfaces. Denotation of events needs to be explicitly included in activity implementations using facilities such as the suspend() function.
38	dtt	–	Not supported.
39	dbr	+	Directly supported via transition conditions and start conditions.

C FLOWer

This evaluation is based on FLOWer 2.05.

Nr	Pattern	Score	Motivation
1	t-d	+/–	Use of restricted option allows update of data element to be limited to single step but other steps can view data value.
2	b-d	+	Each plan can have its own data elements.
3	s-d	+/–	Restricted data elements allow data update to be limited to defined activities.
4	mi-d	+	Fully supported through dynamic plans.
5	c-d	+	Default binding for a data element.
6	wf-d	–	Not supported.
7	e-d	+	Supported via mappings, document types and actions.
8	di-t	+	Supported by shared data elements.
9	di-b-s	+/–	Supported via implicit data passing between shared data elements but no ability to restrict the range of data elements passed.
10	di-s-b	+/–	As for pattern 9.
11	di-a-mi	+	Each instance of a dynamic plan can have discrete data elements which can be passed to/from it.
12	di-mi-a	+	As for pattern 11.
13	di-c-c	+/–	Indirectly supported via external database.
14	di-t-e-push	+	Supported via the Action Operation and Mapping Object (using Out/Out&Insert type) constructs.
15	di-e-t-pull	+	Supported via the mapping construct using In type.
16	di-e-t-push	+/–	Values of data elements in forms associated with tasks can be updated via CHIP library API.
17	di-t-e-pull	+/–	Values of data elements in forms associated with tasks can be queried via CHIP library API.
18	di-c-e-push	+	Supported via the mapping construct using Out/Out&Insert type.
19	di-e-c-pull	+	Supported via the mapping construct using In type.
20	di-e-c-push	+	Supported via CHIP library API.
21	di-c-e-pull	+	Supported via CHIP library API.
22	di-w-e-push	–	Not supported.
23	di-e-w-pull	–	Not supported.
24	di-e-w-push	–	Not supported.
25	di-w-e-pull	–	Not supported.
26	dt-val-in	–	Not supported.
27	dt-val-out	–	Not supported.
28	dt-cico	+/–	Supported via InOut mapping construct.
29	dt-r-unl	+	Standard means of data passing.
30	dt-r-l	+/–	Limited concurrency support. Case can only be updated by one user at a time but others can still view case data.

Nr	Pattern	Score	Motivation
31	dt-in	+/–	Indirectly supported via mappings and derived elements.
32	dt-out	+/–	Indirectly supported via mappings and derived elements.
33	t-pre-de	+	Supported in various ways e.g. milestone defined for a mandatory data element can defer subsequent task initiation until required data element is available.
34	t-pre-dv	+	Milestones can also have a condition delaying invocation of subsequent task until condition is met.
35	t-post-de	+	Achieved by making a data element mandatory. Task will not complete until the data element has a defined value.
36	t-post-dv	+	Modelling elements can have a postcondition. Task will not complete until the postcondition (which should be based on the required parameter value) is met.
37	ett	+	Milestones can wait for data element to have (specific) value set. External processes can directly update value of data elements via CHIP library and cause case execution to resume. CHIP library also provides facilities to create new cases and plans.
38	dtt	+	Directly supported via inclusion of milestones with data-based conditions preceding tasks that are waiting on the data condition.
39	dbr	+/–	Choices can depend on data but only exclusive choice supported.

D COSA

This evaluation is based on COSA 4.2.

Nr	Pattern	Score	Motivation
1	t-d	+	Directly supported by (persistent) ACTIVITY and (non-persistent) STD attributes at the activity level.
2	b-d	+	Workflows can be nested and attributes of the form INSTANCE.name provide a means of supporting block data.
3	s-d	–	Not supported.
4	mi-d	+	Support for distinct data elements in multiply triggered tasks and tasks with shared sub-workflow decompositions.
5	c-d	+	Supported via tool agent INSTANCE (INSTANCE.name).
6	wf-d	+/–	COSA allows for attributes of the form DEFINITION.name however these are fixed at design time.
7	e-d	+	External data elements can be accessed via DDE, ODBC, XML, OS tool agents.
8	di-t	+	Supported by data sharing (implicit data passing).
9	di-b-s	+/–	Supported by data sharing (implicit data passing) but no ability to limit the range of items passed.
10	di-s-b	+/–	As for pattern 9.
11	di-a-mi	–	Not supported.
12	di-mi-a	–	Not supported.
13	di-c-c	+/–	Indirectly supported via external database.
14	di-t-e-push	+	Supported through tool agents e.g. ODBC.
15	di-e-t-pull	+	Supported through tool agents e.g. ODBC.
16	di-e-t-push	+	Supported via triggers which allow data to be passed to a workflow activity.
17	di-t-e-pull	+/–	Achievable using a combination of triggers and tool agents.
18	di-c-e-push	–	Not supported.
19	di-e-c-pull	–	Not supported.
20	di-e-c-push	+	Trigger functions allow process instance and folder data to be set.
21	di-c-e-pull	+	Trigger functions allow process instance and folder data to be read.
22	di-w-e-push	–	Not supported.
23	di-e-w-pull	–	Not supported.
24	di-e-w-push	–	Not supported.
25	di-w-e-pull	–	Not supported.
26	dt-val-in	–	Not supported.
27	dt-val-out	–	Not supported.
28	dt-cico	–	Not supported.

Nr	Pattern	Score	Motivation
29	dt-r-unl	+	Standard means of data passing.
30	dt-r-l	–	Not supported.
31	dt-in	–	Not supported.
32	dt-out	–	Not supported.
33	t-pre-de	+	By specifying transition conditions for tasks based on the CONDITION.TOKEN and STD.exists tool agents, subsequent task invocation can be delayed until required data element is available.
34	t-pre-dv	+	By specifying transition conditions for tasks based on the required data values and CONDITION.TOKEN tool agent, subsequent task invocation can be delayed until required data values are met.
35	t-post-de	–	Not supported.
36	t-post-dv	–	Not supported.
37	ett	+	Triggers provide direct support for all three pattern variants.
38	dtt	+	Directly supported by including a transition condition (which may include CONDITION.TOKEN) corresponding to the required data condition on the incoming arc to the task to be triggered.
39	dbr	+	Supported by conditions on input and output arcs. Both exclusive choice and multi-choice supported.

E XPDL

This evaluation is based on XPDL 1.0.

Nr	Pattern	Score	Motivation
1	t-d	–	Only workflow processes can have data elements.
2	b-d	+	Supported through the block activity construct.
3	s-d	–	Not supported.
4	mi-d	+	Support for distinct data elements in tasks with shared sub-workflow decompositions.
5	c-d	+	A workflow process can have 'data elements'. In case of nesting the scope is not 100% clear.
6	wf-d	+/–	It appears that the 'data fields' of a package can be used for this. Its not clear how the values can be modified.
7	e-d	–	No explicit support.
8	di-t	+	All tasks refer to the same data.
9	di-b-s	+	The element 'subflow' provides actual parameters which should be matched by the formal parameters of the corresponding workflow process. This is not defined out in detail, but the intention is clear.
10	di-s-b	+	As for pattern 9.
11	di-a-mi	–	No support for multiple instances.
12	di-mi-a	–	No support for multiple instances.
13	di-c-c	+/–	Indirectly achievable via coordinated web services operations in sending and receiving cases.
14	di-t-e-push	+	Data elements can be passed to other web services.
15	di-e-t-pull	+	Data can be synchronously requested from other web services.
16	di-e-t-push	–	Not reported.
17	di-t-e-pull	–	Not reported.
18	di-c-e-push	–	Not reported.
18	di-e-c-pull	–	Not reported.
20	di-e-c-push	–	Not reported.
21	di-c-e-pull	–	Not reported.
22	di-w-e-push	–	Not reported.
23	di-e-w-pull	–	Not reported.
24	di-e-w-push	–	Not reported.
25	di-w-e-pull	–	Not reported.
26	dt-val-in	+/–	Although call by value is the standard mechanism (see Section 7.1.2.1), there is not explicit data passing between activities. However, the element 'subflow' provides actual parameters which should be matched by the formal parameters of the corresponding workflow process.

Nr	Pattern	Score	Motivation
27	dt-val-out	+/–	As for 26.
28	dt-cico	+/–	As for 26.
29	dt-r-unl	+	Directly supported where data is passed implicitly by shared variables.
30	dt-r-l	–	No support, see Section 7.1.2.1 of WFMC-TC-1025 (Oct. 2002).
31	dt-in	–	Not mentioned.
32	dt-out	–	Not mentioned.
33	t-pre-de	–	Not supported.
34	t-pre-dv	+	Inclusion of additional edge around task with transition condition that is the negation of required data values allows task to be skipped when values not met.
35	t-post-de	–	Not supported.
36	t-post-dv	–	Not supported.
37	ett	–	Not mentioned.
38	dtt	–	Not supported.
39	dbr	+	Through transition conditions and transition restrictions. Both exclusive choice and multi-choice supported.

F BPEL4WS

This evaluation is based on BPEL4WS version 1.1.

Nr	Pattern	Score	Motivation
1	t-d	+/–	Scopes provide a means of limiting the visibility of variables to smaller blocks approaching task level binding.
2	b-d	–	Sub-processes are not supported.
3	s-d	+	Directly supported by scope construct.
4	mi-d	–	Not supported. Data elements are scoped at case level.
5	c-d	+	The default scoping for variables is process level.
6	wf-d	–	Not supported.
7	e-d	+	Accessing to environment data via web service calls.
8	di-t	+	Data elements can be passed between activities in messages or via shared variables.
9	di-b-s	–	Not supported.
10	di-s-b	–	Not supported.
11	di-a-mi	–	Not supported.
12	di-mi-a	–	Not supported.
13	di-c-c	+/–	Indirectly achievable using the invoke construct.
14	di-t-e-push	+	The invoke construct supports the passing of data elements to other web services on an asynchronous basis.
15	di-e-t-pull	+	The invoke construct also supports the request of data elements from other web services on a synchronous basis.
16	di-e-t-push	+/–	Indirectly achievable via the receive construct or event handlers although the data update operation requires specific coding.
17	di-t-e-pull	+/–	Indirectly achievable via the receive/reply constructs or event handlers although the operation requires specific coding.
18	di-c-e-push	–	Not supported.
19	di-e-c-pull	–	Not supported.
20	di-e-c-push	–	Not supported.
21	di-c-e-pull	–	Not supported.
22	di-w-e-push	–	Not supported.
23	di-e-w-pull	–	Not supported.
24	di-e-w-push	–	Not supported.
25	di-w-e-pull	–	Not supported.
26	dt-val-in	+	Option to pass data elements between activities by value (using messages).
27	dt-val-out	+	Option to pass data elements between activities by value (using messages).
28	dt-cico	–	Not supported.
29	dt-r-unl	+	No concurrency control is applied by default.

Nr	Pattern	Score	Motivation
30	dt-r-l	+/–	Some measure of concurrency control can be achieved through the use of serializable scopes.
31	dt-in	–	Not supported.
32	dt-out	–	Not supported.
33	t-pre-de	+/–	Can be indirectly facilitated via exception handlers dealing with attempts to use uninitialised parameters but requires dedicated scope for each activity.
34	t-pre-dv	+	Inclusion of additional link around task with transition condition that is the negation the required data values allows task to be skipped when data values not met.
35	t-post-de	–	Not supported.
36	t-post-dv	–	Not supported.
37	ett	+	Direct event support via event handlers and the receive construct. New process instances can also be invoked externally.
38	dtt	+/–	Indirectly achievable by including a data condition monitoring task in the process and triggering the to-be-triggered task via a link or alternatively using message events for triggering.
39	dbr	+	Both exclusive choice and multi-choice supported.

G Evaluation Criteria

The following tables summarise the evaluation criteria for each of the workflow data patterns.

- To achieve a given + rating, a workflow engine must demonstrate that it complies with each of the criteria specified.
- To achieve a +/− rating it must satisfy at least one of the criteria listed.
- Otherwise a − rating is recorded.

Nr	Pattern	+ Rating	+/− Rating
1	Task Data	① Data can be explicitly declared at task level with task level scoping ② Data support exists within the workflow tool	① Task data can be indirectly specified via a programmatic implementation of a task ② Task data can be specified at task level but is visible (although not mutable) outside the task ③ Facilities do not exist to ensure data elements are initialised for each task instance
2	Block Data	① Data can be explicitly declared at block task level with block task level scoping ② Data support exists within the workflow tool ③ Facilities exist for formal parameter passing to and from a block	① Block data can be indirectly specified via a programmatic implementation of a block task or sub-workflow
3	Scope data	① Direct workflow support for binding data elements to a subset of the task instances in the workflow process	① Ability to limit data update or read actions to defined tasks. ② Achievable via programmatic extensions
4	Multiple Instance Data	① Direct tool support for multiple execution instances of the same task and/or shared block task implementation and/or multiple task triggering	① Later instances of the same task retain data values from the execution of an earlier instance

Nr	Pattern	+ Rating	+/− Rating
		② Data elements can be defined and scoped to individual task instances	② Data isolation between multiple task instances can be achieved through programmatic extensions
5	Case Data	① Direct tool support for data elements at case level with case level scoping ② Case data visible to all components of the case	① Effect of case level data sharing can be achieved through programmatic extensions Data elements must be passed to sub-workflows
6	Workflow Data	① Workflow data visible to all components of a workflow ② Direct tool support for workflow level data	① Effect of workflow level data sharing can be achieved through programmatic extensions Unable to update workflow data elements during execution
7	Environment Data	① Workflow tool provides direct support for accessing external data elements both from files and via external applications or APIs ② No limitations on source or format of data elements that can be accessed	① Access to external data elements can be achieved through programmatic extensions
8	Data Interaction between Tasks	① Data elements can be directly passed from one task to another	① Data Interaction involves some form of intermediation or can only occur via shared data repositories
9	Data Interaction - Block Task to Sub-workflow Decomposition	① Data elements available to a block task are able to be passed to or are accessible in the associated sub-workflow ② There is some degree of control over which elements at block task level are made accessible in the sub-workflow	① The range of elements accessible within the sub-workflow is more limited than those available in the block task
10	Data Interaction - Sub-workflow Decomposition to Block Task	① Data elements at sub-workflow level can be passed to or made accessible in the corresponding block task	① There are limited facilities for controlling which sub-workflow data elements can be made available to the block task
11	Data Interaction to Multiple Instance Task	① Multiple instance tasks directly supported ② Data elements can be passed from an atomic task to all instances of a multiple instance task	① Multiple instances not directly supported ② Programmatic intervention required to facilitate data passing

Nr	Pattern	+ Rating	+/− Rating
		③ Workflow handles synchronisation of data passing and any necessary data replication ④ Facilities are available to allocate sections of an aggregate data element to specific task instances ⑤ Data elements in each task instance are independent of those in other task instances	③ No facilities for refreshing/resetting data elements for re-entrant task instances
12	Data Interaction from Multiple Instance Task	① Multiple instance tasks directly supported ② Data elements can be aggregated from multiple task instances and forwarded to subsequent task instance(s) ③ Workflow handles synchronisation of data passing and any necessary data replication	① Multiple instances not directly supported ② Programmatic intervention required to facilitate data passing
13	Data Interaction - Case to Case	① Data elements can be directly passed between workflow cases	① Interaction occurs via an external data repository ② Not directly achievable or programmatic intervention required
14	Data Interaction - Task to Environment - Push-Oriented	① Direct workflow support for task-initiated data passing to external applications	① Limitations on the type of data elements that can be passed ② Achievable via programmatic extensions
15	Data Interaction - Environment to Task - Pull-Oriented	① Direct workflow support for task instances to initiate requests for data and receive replies from external applications	① Achievable via programmatic extensions
16	Data Interaction - Environment to Task - Push-Oriented	① Direct workflow support for task instances to accept externally generated data during execution	① Achievable via programmatic extensions
17	Data Interaction - Task to Environment - Pull-Oriented	① Direct workflow support for tasks to respond to external requests for data	① Achievable via programmatic extensions
18	Data Interaction - Case to Environment - Push-Oriented	① Direct workflow support for case-initiated data passing to external applications at any time	① Achievable via programmatic extensions

Nr	Pattern	+ Rating	+/− Rating
19	Data Interaction - Environment to Case - Pull-Oriented	① Direct workflow support for cases to initiate requests for data and receive replies from external applications at any time	① Achievable via programmatic extensions
20	Data Interaction - Environment to Case - Push-Oriented	① Direct workflow support for cases to accept externally generated data at any time during execution	① Achievable via programmatic extensions
21	Data Interaction - Case to Environment - Pull-Oriented	① Direct workflow support for cases to respond to external requests for data at any time	① Achievable via programmatic extensions
22	Data Interaction - Workflow to Environment - Push-Oriented	Direct support for the workflow engine to initiate data passing to external applications	① Achievable via programmatic extensions
23	Data Interaction - Environment to Workflow - Pull-Oriented	① Direct support for the workflow engine to initiate requests for data and receive replies from external applications	① Achievable via programmatic extensions
24	Data Interaction - Environment to Workflow - Push-Oriented	① Direct support for the workflow engine to accept externally generated data during execution	① Achievable via programmatic extensions
25	Data Interaction - Workflow to Environment - Pull-Oriented	① Direct workflow support for workflows to respond to external requests for data	① Achievable via programmatic extensions
26	Data Passing by Value - Incoming	① Workflow components are able to accept data elements passed to them as values	① Achievable via programmatic extensions
27	Data Passing by Value - Outgoing	① Workflow components are able to pass data elements to subsequent components by value	① Achievable via programmatic extensions
28	Data Passing - Copy In/Copy Out	① Direct workflow support exists for a workflow component to copy in data elements from an external source (either within or outside the workflow) when commencing execution and to copy the (possibly updated) values back to the external source at the conclusion of execution	① Indirectly achievable via programmatic extensions

Nr	Pattern	+ Rating	+/− Rating
29	Data Passing by Reference - Unlocked	① Block, case or workflow level data can be accessed by all associated workflow components ② Programmatic extensions required	① Restricted data visibility limits the use of this strategy
30	Data Passing by Reference - Locked	① Block, case or workflow level data can be accessed by all workflow components ② Locking/concurrency control facilities are provided	① Restricted data visibility limits the use of this strategy ② Programmatic extensions required
31	Data Transformation - Input	① Direct workflow support ② Transformation functions can be specified on all data elements	① Only achievable via programmatic extensions
32	Data Transformation - Output	① Direct workflow support ② Transformation functions can be specified on all data elements	① Only achievable via programmatic extensions ② Limited transformation capability or range
33	Task Precondition - Data Existence	① Direct precondition support for evaluating data element existence at task instance level	① Programmatic extensions required
34	Task Precondition - Data Value	① Direct precondition support at task instance level ② Support for a broad range of preconditions	① Limited range of preconditions supported ② Programmatic extensions required
35	Task Postcondition - Data Existence	① Direct postcondition support for evaluating data element existence at task level ② Must support internal task repetition	① Limited range of postconditions ② Programmatic extensions required
36	Task Postcondition - Data Value	① Direct postcondition support at task level ② Must support internal task repetition	① Limited range of postconditions supported ② Programmatic extensions required
37	Event-based Task Trigger	① Direct workflow support for task instance initiation/suspension on an event trigger ② Provision of facilities for external processes to identify the correct task instance to signal	① Programmatic extensions required
38	Data-based Task Trigger	① Direct workflow support for monitoring conditions based on data elements	① Limited range of potential data triggers

Nr	Pattern	+ Rating	+/− Rating
		② Specific task instances can be initiated when trigger condition is achieved ③ Any data element accessible at case level can serve as a trigger	② Programmatic extensions required
39	Data-based Routing	① Any data element accessible at case level can be utilised in a routing construct ② Direct workflow support ③ Support for both exclusive choice and multi-choice constructs	① Limited range of data elements can be used in routing constructs ② Limited range of routing constructs supported ③ Programmatic extensions required

Table 7: Data Pattern Evaluation Criteria for Workflow Systems