Workflow Data Patterns: Identification, Representation and Tool Support*

Nick Russell[1], Arthur H.M. ter Hofstede[1], David Edmond[1], and Wil M.P. van der Aalst[1,2]

[1] School of Information Systems, Queensland University of Technology, GPO Box 2434, Brisbane QLD 4001, Australia
{n.russell, a.terhofstede, d.edmond}@qut.edu.au
[2] Department of Technology Management, Eindhoven University of Technology, P.O.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. 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, workflow standards and web-service composition languages.

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.

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

Detailed examination of a number of workflow tools and business process modelling paradigms suggests that the way in which data is structured and utilised within these tools has a number of common characteristics. Indeed these characteristics bear striking similarities to the *Workflow Patterns* [2] and *Design Patterns* [5] initiatives in terms of their generic applicability, except in this case they refer specifically to the data perspective [7] 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 constructs 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 this has fuelled the development of a multitude of both commercial workflow engines and research prototypes each with unique features and capabilities. Despite attempts by industry bodies such as the Workflow Management Coalition (`www.wfmc.org`) and the Object Management Group (`www.omg.org`) to provide standards for workflow management systems, there is limited adoption by commercial vendors. Perhaps the most notable 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 [1], Event-Driven Process Chains (EPCs) [11] and UML Activity Diagrams [4] although none of these have achieved broad usage.

Data modelling in the context of workflow systems is an area that has received particularly scant attention. Many of the significant workflow modelling proposals and prototypes take a uniform approach to data representation and utilisation. In ADEPT [9], all data elements are assumed to be represented by global workflow variables. MENTOR [13] proposes that data flow is modelled via Activity Diagrams linked to the control-flow perspective represented in the form of State Charts. WASA [8] assumes that data is characterised in the form of data containers which are passed between workflow activities although the later WASA2 project [12] supports a more varied range of data objects by leveraging underlying CORBA services. INCAs [3] is one of the few initiatives which proposes a broader range of data representations and interaction facilities.

One of the major impediments to a generic modelling technique is the broad range of offerings that fall under the "workflow umbrella" [6] – 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 [2] has taken an empirical approach to identifying the major control-flow 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. It is interesting to note that this work has directly influenced tool selection processes, commercial and open-source workflow systems, and workflow standards (see `www.workflowpatterns.com` for details). This paper adopts an approach similar to that in [2] although in this case, the focus is on the data perspective. 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. It identifies 40 data patterns that recur in workflow systems and describes a selection of these in detail[1]. It also examines their use across six major workflow products, standards and web service composition languages.

2 Workflow and Data Concepts

2.1 Workflow Structure

Before we describe the data perspective 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. 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. A *task* corresponds to a single unit of work. Four distinct types of task are denoted: *atomic*, *block*, *multiple-instance* and *multiple-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. 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*

[1] Readers seeking a comprehensive description of all 40 patterns are referred to [10].

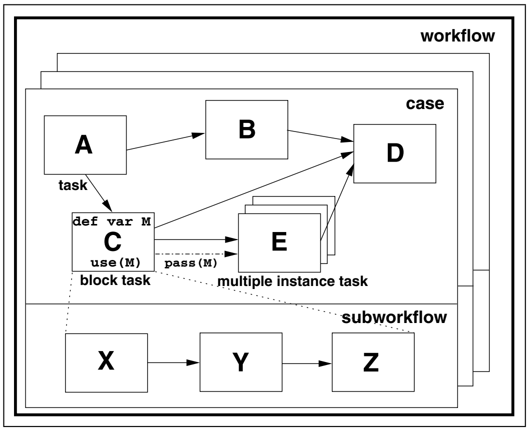

Fig. 1. Components of a workflow

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 *multiple-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 which is illustrated with a broken (dash-dot) line. The control and data channels may be combined. 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.

2.2 Data Characterisation

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-flow perspective.

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*, which relate to the definition and scope of data elements and the manner in which they can be utilised by various components of a workflow process; *data interaction*, which focus on the manner in which data is communicated between active elements within and outside of 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; and *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.

These characteristics are examined in detail in Section 3 and between them, they form the basis for the identification of 40 patterns relevant to the data perspective of PAIS. As validation of the broad applicability of the patterns identified, their occurrence is examined in three major workflow engines (Staffware, WebSphere MQ and COSA), a case handling system (FLOWer), a web service composition language (BPEL4WS) and a workflow standard (XPDL). The results of these evaluations are presented in Section 4.

3 Data Patterns

3.1 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 individual data elements are bound to a specific workflow construct (e.g. a task or a block) and this binding defines the scope in which the data element can be accessed. More generally, it also influences 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. Eight data visibility patterns have been identified. The second of these, Block Data, is discussed subsequently in further depth. Lines 1 to 8 of Table 1 provide a complete listing of the data visibility patterns.

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.

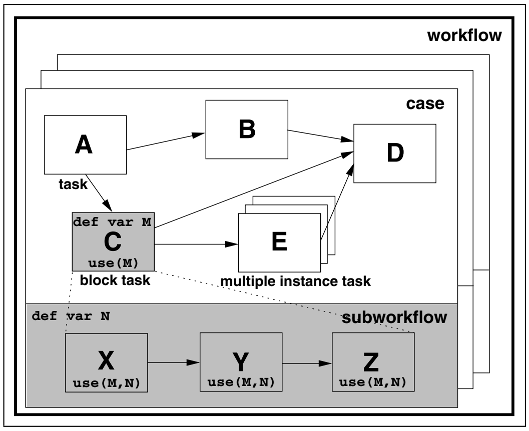

Fig. 2. Block level data visibility

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 2 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.

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[2] 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 WebSphere MQ, 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 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

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

if X were also a block task would this data element also be accessible to task instances in the sub-workflow 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[3], however updates to data elements in sub-workflows are not automatically propagated back to the parent task.

3.2 Data Interaction

Data interaction patterns capture 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 component with some form of information resource or service that operates outside of the context of the workflow engine (i.e. in the external environment).

In total, 18 data interaction patterns have been identified – six of these relate to data interaction between internal workflow components and the remaining twelve describe the various situations where data interaction can occur between a workflow component and the external environment. The complete listing of these patterns is presented in lines 9 to 26 of Table 1. In this section, we describe two internal and one external data interaction patterns in detail.

Pattern 9 (Data Interaction between Tasks)

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 examined 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 3.

[3] Although more than four levels of nesting are not recommended.

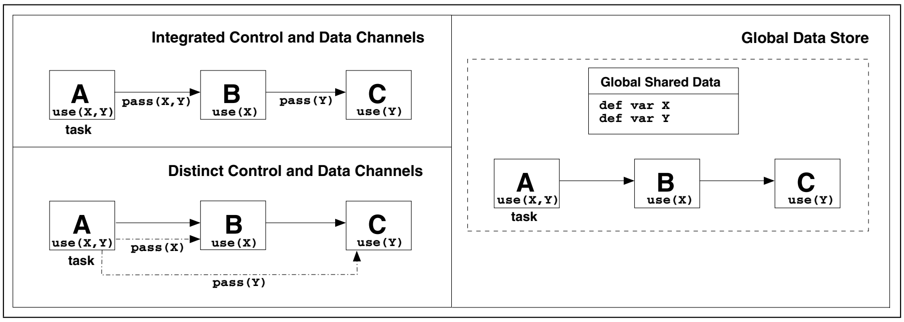

Fig. 3. Approaches to data interaction between tasks

- *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 task commencement.
- *Global data store* – where tasks share the same data elements (typically via access to globally shared data) and no explicit data passing is required. 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.

Most of the 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. WebSphere MQ 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 4. 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 12 (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 *New Albums List* is passed to the *Review Album* task and one task instance is started for each entry on the list. 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 4. 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 specific approaches are as follows:

– *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

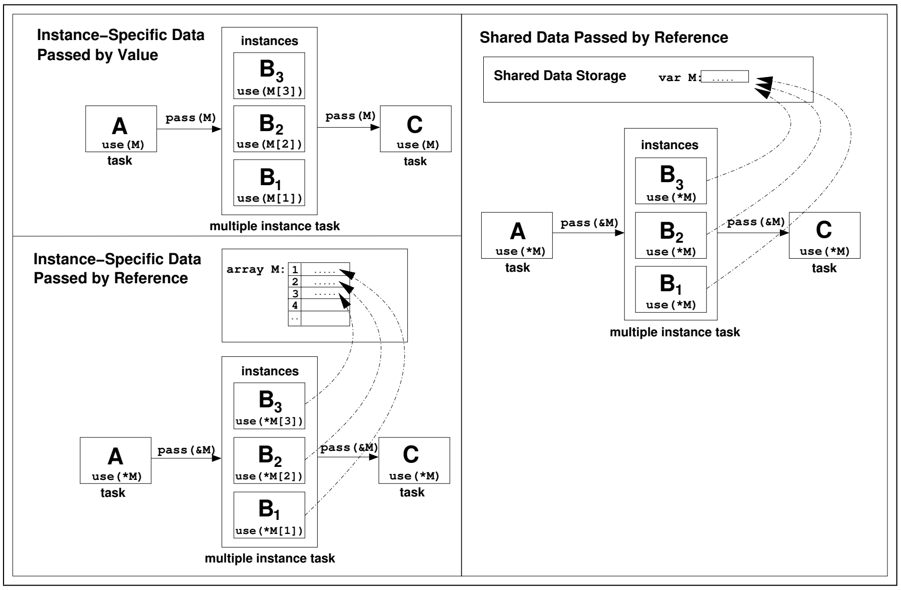

Fig. 4. Data interaction approaches for multiple instance tasks

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 workflow component).

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 16 (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. 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 and *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, CORBA or JMS.

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 (dynamic data exchange) 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. In contrast, WebSphere MQ does not provide any facilities for external integration and requires the underlying programs that implement workflow tasks to provide these capabilities where they are needed.

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. 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, necessitating the workflow to provide notification facilities.

3.3 Data Transfer Patterns

Data transfer patterns focus on 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 3.2 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 seven distinct patterns as listed in lines 27 to 33 of Table 1. In this section, we describe one pattern in detail – Data transfer by value - incoming.

Pattern 27 (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.

WebSphere MQ 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 sub-workflow. As all data elements are case level, there is no explicit data passing between tasks.

3.4 Data-Based Routing

Whereas other groups of patterns focus on characteristics of data elements in isolation from other workflow perspectives (i.e. control, resource, organisational etc.), data-based routing patterns aim to capture the various ways in which data elements can interact with other perspectives and influence the overall operation of the workflow. Constructs such as pre-conditions, post-conditions, triggers and splits are characterised by these patterns and seven of them have been identified as listed in lines 34 to 40 of Table 1. We do not discuss these patterns in detail here and interested readers are referred to [10] for more details.

4 Evaluation of Existing Workflow Products

This section presents the results of a detailed evaluation of support for the 40 workflow data patterns by six workflow systems, standards and web service composition 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" [6]. Specific tools and languages evaluated were Staffware Process Suite v9, WebSphere MQ Workflow 3.4, FLOWer 3.0, COSA 4.2, XPDL 1.0 and BPEL4WS 1.1. A three point assessment scale is used with "+" indicating direct support for the pattern, "+/–" indicating partial support and "–" indicating that the pattern is not implemented. Specific rating criteria have been devised and are detailed in [10].

Lines 1 to 8 indicate the various levels of data construct visibility supported within the tools. 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. Lines 9 to 14 list the results for internal data passing. All offerings supported task-to-task interaction and block task-to-sub-workflow interaction[4]. 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. Lines 15 to 26 indicate the ability of the workflow products to integrate with data sources and applications in the operating environment. WebSphere MQ, 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. Lines 27 to 33 illustrate the mechanisms

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

Table 1. Support for Data Patterns in Workflow Systems

Nr	Pattern	Staffware	WebSphere	FLOWer	COSA	XPDL	BPEL4WS
1	Task Data	–	+/–	+/–	+	–	+/–
2	Block Data	+	+	+	+	+	–
3	Scope Data	–	–	+/–	–	–	+
4	Folder Data	–	–	–	+	–	–
5	Multiple Instance Data	+/–	+	+	+	+	–
6	Case Data	+/–	+	+	+	+	+
7	Workflow Data	+	+	–	+/–	+/–	–
8	Environment Data	+	+/–	+	+	–	+
9	Data Interaction between Tasks	+	+	+	+	+	+
10	Data Interaction – Block Task to Sub-workflow	+	+	+/–	+/–	+	–
11	Data Interaction – Sub-workflow to Block Task	+	+	+/–	+/–	+	–
12	Data Interaction – to Multiple Instance Task	–	–	+	–	–	–
13	Data Interaction – from Multiple Instance Task	–	–	+	–	–	–
14	Data Interaction – Case to Case	+/–	+/–	+/–	+	+/–	+/–
15	Data Interaction – Task to Env. – Push	+	+/–	+	+	+	+
16	Data Interaction – Env. to Task – Pull	+	+/–	+	+	+	+
17	Data Interaction – Env. to Task – Push	+/–	+/–	+/–	+	–	+/–
18	Data Interaction – Task to Env. – Pull	+/–	+/–	+/–	+	–	+/–
19	Data Interaction – Case to Env. – Push	–	–	+	–	–	–
20	Data Interaction – Env. to Case – Pull	–	–	+	–	–	–
21	Data Interaction – Env. to Case – Push	+/–	+/–	+	+	–	–
22	Data Interaction – Case to Env. – Pull	–	–	+	+	–	–
23	Data Interaction – Workflow to Env. – Push	–	+/–	–	–	–	–
24	Data Interaction – Env. to Workflow – Pull	+/–	–	–	–	–	–
25	Data Interaction – Env. to Workflow – Push	–	+/–	–	–	–	–
26	Data Interaction – Workflow to Env. – Pull	+	+	–	+	–	–
27	Data Transfer by Value – Incoming	–	+	–	+/–	+/–	+
28	Data Transfer by Value – Outgoing	–	+	–	+/–	+/–	+
29	Data Transfer – Copy In/Copy Out	–	–	+/–	–	+/–	–
30	Data Transfer by Reference – Unlocked	+	–	+	+	+	+
31	Data Transfer by Reference – Locked	–	–	+/–	–	–	+/–
32	Data Transformation – Input	+/–	–	+/–	–	–	–
33	Data Transformation – Output	+/–	–	+/–	–	–	–
34	Task Precondition – Data Existence	+	–	+	+	–	+/–
35	Task Precondition – Data Value	+	–	+	+	+	+
36	Task Postcondition – Data Existence	+/–	+	+	–	–	–
37	Task Postcondition – Data Value	+/–	+	+	–	–	–
38	Event-based Task Trigger	+	+/–	+	+	–	+
39	Data-based Task Trigger	–	–	+	+	–	+/–
40	Data-based Routing	+/–	+	+/–	+	+	+

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 is limited. Lines 34 to 40 indicate the ability of the data perspective to influence the control perspective within each product. FLOWer demonstrates outstanding capability in this area and Staffware, WebSphere MQ 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 trigger support. BPEL4WS would also benefit from better pre and postcondition support and lacks data-based triggering.

Through these evaluations a number of insights have been gained in relation to the current level of data support in workflow systems. Current workflow modelling techniques centre on the capture of control-flow and offer minimal support for documenting data requirements. This difficulty extends into workflow design tools which typically provide fragmented facilities for incorporating data requirements in workflows. There is little support for multiple instance tasks in current tools (which provide for true task parallelism) and where it exists, the level of data support is minimal. There also appears to have been little learnt from the database field and the evaluations revealed limited support for data persistence and concurrency handling in the tools examined.

5 Conclusion

This paper has identified 40 new *workflow data patterns* which describe the manner in which data is defined and utilised in workflow systems. The main contribution of this work is that it is the first systematic attempt to provide a taxonomy of data usage in workflow systems in a technology-independent manner. Validation of the applicability of these patterns has been achieved through a detailed review of six workflow systems, standards and web service composition languages. The results of this review indicate that the data patterns identified are applicable not only to workflow systems but that they are also of relevance to process-aware information systems more generally.

Evaluation of pattern support in current tools gives a valuable insight into the operation of workflow systems and the data patterns identified have a number of practical uses. First, they provide an effective foundation for training workflow designers and developers. Second, they provide a means of assessing tool capabilities and are particularly useful in tool evaluation and selection exercises (e.g. tender evaluations). Finally, they offer the basis for vendors to identify functionality gaps and potential areas for enhancement.

References

1. W.M.P. van der Aalst. The Application of Petri Nets to Workflow Management. *Journal of Circuits, Systems and Computers*, 8(1):21–66, 1998.
2. W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, and A.P. Barros. Workflow Patterns. *Distributed and Parallel Databases*, 14(3):5–51, 2003.
3. D. Barbara, S. Mehrotra, and M. Rusinkiewicz. INCAs: Managing Dynamic Workflows in Distributed Environments. *Journal of Database Management*, 7(1):5–15, 1996.
4. M. Dumas and A. ter Hofstede. UML Activity Diagrams as a Workflow Specification Language. In M. Gogolla and C. Kobryn, editors, *Proceedings of the Fourth International Conference on the Unified Modeling Language (UML 2001)*, LNCS 2185, pages 76–90, Toronto, Canada, 2001. Springer.
5. E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley, Boston, USA, 1995.
6. 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.
7. S. Jablonski and C. Bussler. *Workflow Management: Modeling Concepts, Architecture and Implementation.* Thomson Computer Press, London, UK, 1996.
8. C.B. Medeiros, G. Vossen, and M. Weske. WASA: A Workflow-Based Architecture to Support Scientific Database Applications. In N. Revell and A.M. Tjoa, editors, *Proceedings of the 6th International Workshop and Conference on Database and Expert Systems Applications (DEXA)*, pages 574–583, London, UK, 1995. Springer.
9. M. Reichert and P. Dadam. ADEPTflex - Supporting Dynamic Changes of Workflows Without Losing Control. *Journal of Intelligent Information Systems*, 10(2):93–129, 1998.
10. N. Russell, A.H.M. ter Hofstede, D. Edmond, and W.M.P. van der Aalst. Workflow Data Patterns (Revised Version). Technical Report FIT-TR-2004-01, Queensland University of Technology, Brisbane, Australia, 2004. `http://www.bpmcenter.org`.
11. A.-W. Scheer. *ARIS - Business Process Modelling.* Springer, Berlin, Germany, 2000.
12. G. Vossen and M. Weske. The WASA2 Object-Oriented Workflow Management System. In A. Delis, C. Faloutsos, and S. Ghandeharizadeh, editors, *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD 1999)*, pages 587–589, Philadelphia, Pennsylvania, USA, 1999. ACM Press.
13. D. Wodtke, J. Weissenfels, G. Weikum, and A. Kotz-Dittrich. The Mentor Project: Steps Towards Enterprise-Wide Workflow Management. In S.Y.W. Su, editor, *Proceedings of the 12th International Conference on Data Engineering (ICDE 1996)*, pages 556–565, New Orleans, Louisiana, USA, 1996. IEEE Computer Society.