# On the Suitability of UML 2.0 Activity Diagrams for Business Process Modelling*

**Nick Russell**[1] **Wil M.P. van der Aalst**[2,1] **Arthur H.M. ter Hofstede**[1]
**Petia Wohed**[3]

[1]School of Information Systems, Queensland University of Technology
GPO Box 2434, Brisbane QLD 4001, Australia
{n.russell, a.terhofstede}@qut.edu.au

[2]Department of Technology Management, Eindhoven University of Technology
GPO Box 513, NL5600 MB Eindhoven, The Netherlands
w.m.p.v.d.aalst@tm.tue.nl

[3]Department of Computer and Systems Sciences, Stockholm University/KTH
Forum 100, 164 40 Kista, Sweden
petia@dsv.su.se

**Abstract**

UML is posited as the "swiss army knife" for systems modelling and design activities. It embodies a number of modelling formalisms that have broad applicability in capturing both the static and dynamic aspects of software systems. One area of UML that has received particular attention is that of Activity Diagrams (ADs), which provide a high-level means of modelling dynamic system behaviour. In this paper we examine the suitability of UML 2.0 Activity Diagrams for business process modelling, using the Workflow Patterns as an evaluation framework. The Workflow Patterns are a collection of patterns developed for assessing control-flow, data and resource capabilities in the area of Process Aware Information Systems (PAIS). In doing so, we provide a comprehensive evaluation of the capabilities of UML 2.0 ADs, and their strengths and weaknesses when utilised for business process modelling.

## 1 Introduction

The Unified Modeling Language (UML) is increasingly being seen as the *de-facto* standard for software modelling and design. The most recent version (2.0) (OMG 2004) includes 13 distinct modelling notations ranging from high-level use case diagrams, which depict the interactions and relationships between (human) actors and major business functions, through to low-level object diagrams which capture instances of individual data objects, their constituent data elements and values, and their relationships with other data objects.

The various modelling notations essentially divide into three main groups:

- Behaviour diagrams, which describe the overall functionality of the software at a relatively high level of abstraction;
- Interaction diagrams, which further augment the behaviour diagrams and present a more detailed description of system functionality in terms of object interactions; and
- Structure diagrams, which capture the underlying static structure of a software system at various levels from individual objects to overall application packages.

At its heart, UML is an object-oriented set of notations and is particularly effective for capturing detailed design models of software systems in a form which is suitable for translation into some form of enactment technology (usually program code) either by suitably qualified developers or in a semi-automated manner via CASE technology. However, the breadth of UML ensures that it also has potential applicability in a number of other scenarios such as *business process modelling* although in such a distinct domain, some notations (e.g. the class of behaviour diagrams) are likely to be more useful than others.

Over the past decade, the economics of software usage have begun to change, and it is increasingly common for new systems to be based on modifications of widely distributed software products rather than being custom software developments. There is also a broader view being taken as to the scope in which a system operates and the recognition that true cost-benefits can only occur when software processes are aligned with organisational processes. These notions have been reinforced by the advent of organisation-wide software packages such as Enterprise Resource Planning (ERP), Customer Relationship Management (CRM) and Workflow Management (WFM) systems which bind together multiple operational groups within an organisation in a set of integrated business processes.

As a consequence of this shift, there is an increased interest in software process modelling in an organisational context – generally termed *business process modelling* or *enterprise modelling*, depending on whether the focus of the modelling is on the business process or the overall organisation. Several modelling techniques have been proposed as an all-encompassing standard for this role, however no one formalism is predominant in this area (Mendling, Neumann & Nüttgens 2005). One of the major reasons cited for this (zur Muehlen & Rosemann 2004) is the wide disparity in the needs and viewpoints of various modellers and designers.

In this paper, we investigate the *suitability* of Activity Diagrams for business process modelling. This notation is the most detailed form of process modelling within UML. However, its applicability to the

*This work is funded in part by ARC Discovery Project DP0451092.

business process modelling domain in a general sense is not immediately evident and merits more detailed examination in order to determine what advantages it offers and what its shortcomings are.

We base this evaluation on the Workflow Patterns[1], a taxonomy of generic, recurring constructs originally devised to evaluate workflow systems, and more recently used to successfully evaluate workflow standards, business process languages and process-aware information systems in general (Dumas & ter Hofstede 2001, White 2004, Wohed, Perjons, Dumas & ter Hofstede 2003). In accordance with Jablonski and Bussler's original classification (Jablonski & Bussler 1996), these patterns span the control-flow, data and resource perspectives of PAIS. Our choice of this evaluation framework is based on the fact that it is 1) widely used, 2) well accepted, 3) comprehensible to the IT practitioner, 4) at a sufficiently detailed level of abstraction to provide a comprehensive basis for assessing their capabilities of business process modelling languages and 5) the most complete and powerful framework for this form of assessment currently in existence.

The main contributions of this paper are as follows:

- It is the first multi-perspective evaluation of the expressive capabilities of UML 2.0 ADs;
- It provides an assessment of the overall suitability of UML 2.0 ADs for business process modelling; and
- It identifies several areas of potential improvements to UML 2.0 ADs to further strengthen their use for this purpose.

This paper focuses on the the new version of UML Activity Diagrams (ADs) 2.0 (OMG 2004), which differ considerably from their precursor UML 1.4 ADs.[2] Previous evaluations (cf. (Dumas & ter Hofstede 2001, Opdahl & Henderson-Sellers 2002)) have analysed the expressive power of UML 1.4 ADs. There has also been a review of the capabilities of the control-flow perspective of UML 2.0 ADs (White 2004), however the limited focus of this investigation has restricted its usefulness as a means of assessing their overall suitability for general modelling purposes.

The remainder of this paper proceeds as follows: Sections 2 and 3 provide an overview of business process modelling languages and UML 2.0 ADs respectively. Sections 4, 5 and 6 present evaluations of the control-flow, data and resource perspectives of UML 2.0 ADs. Section 7 reviews the suitability of UML 2.0 ADs for business process modelling in light of the findings in the preceding sections. It also offers some recommendations for further improving their capabilities in this area.

## 2 Business Process Modelling Languages

Business process modelling is essentially a convergence of two related modelling domains: *process modelling* (cf. (Curtis, Kellner & Over 1992, Rolland 1997, Rolland 1998)) which seeks to provide "an abstract representation of a process architecture, design, or definition" ((Humphrey & Feiler 1992), p.33) and *enterprise modelling* or *business modelling* which focuses on documenting an organisation from a holistic standpoint, capturing details of its overall purpose and goals in addition to more concrete details such as organisational structure and significant business activities (cf. (Vernadat 1996, Bubenko, Persson & Stirna 2001, Eriksson & Penker 2000, Marshall 1999)).

Whilst there is significant overlap between them, they are generally viewed as having distinct motivations (Jablonski & Bussler 1996), and this is best exemplified by the formalisms used for modelling in the two areas. Process models are usually based on a single technique, such as Data Flow Diagrams (DFDs), Event-driven Process Chains (EPCs), UML Activity Diagrams or Petri-Nets, which is used to capture the details of the process in question. In contrast, enterprise modelling generally requires a range of modelling techniques to be used in conjunction with each other in order to capture the required domain information. This tends to favour the use of integrated suites of modelling techniques such as ARIS (Scheer 2000), UML (Eriksson & Penker 2000, Marshall 1999) and EKD (Bubenko et al. 2001) which possess a sufficiently broad range of integrated modelling formalisms.

Business process modelling essentially seeks to provide a detailed description of a business process in an organisational context. There are a range of potential modelling languages that can be used for this purpose and Kueng *et. al.* (Kueng, Kawalek & Bichler 1996) have proposed a taxonomy of business process modelling techniques which classifies them into four groups:

- *Activity-oriented approaches* - focusing on the definition of business processes as a sequence of activities;
- *Object-oriented approaches* - leveraging the more comprehensive modelling constructs of object-orientation to capture business processes;
- *Role-oriented approaches* - modelling business processes based on the specific organisational roles and responsibilities involved; and
- *Speech-act approaches* - viewing business processes in the context of the speech-act language action paradigm.

Other considerations for the selection of an appropriate business process modelling language to use in a particular scenario include the *kind* of process being modelled and the *purpose* for which the modelling is being undertaken. Rolland (1998) classifies processes into three kinds: *strategic* – which investigate alternative ways of achieving a required outcome and produce a plan for doing so; *tactical* – which focus on the tactics for achieving the plan; and *implementation* – which describe how the plan will be achieved. Similarly, individual process models may be developed with one of three possible aims: *descriptive* – which describe what actually happens during a process; *prescriptive* – which define how a process might or should be performed; and *explanatory* – which detail the rationale for a process and link it to the requirements it must fulfill.

## 3 UML 2.0 Activity Diagrams

In UML Activity Diagrams the fundamental unit of behaviour specification is the Action. "An action takes a set of inputs and converts them to a set of outputs, though either or both sets may be empty" ((OMG 2004), p.229). Actions may also modify the state of the system. The language provides a very detailed action taxonomy, identifying more than 40 different action types in detail. A comprehensive

[1] See `www.workflowpatterns.com` for comprehensive details.

[2] The semantics of UML 2.0 ADs are based on token flow instead of statecharts as in UML 1.4.

discussion of them is beyond the scope of this paper and in Figure 1a we only present the action types that we have found to be most relevant to our evaluations. These are the generic Action concept, Accept Event, Send Signal, and Call Behavior Action constructs.

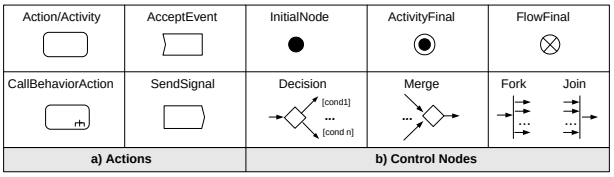


Figure 1: The main constructs in UML 2.0 ADs

In order to represent the overall behaviour of a system, the concept of the Activity is used. Activities are composed of actions and/or other activities and they define dependencies between their elements. Graphically, they are composed of nodes and edges. The edges connect the nodes in sequential order. Nodes represent either Actions, Activities, Data Objects, or control nodes. The various types of control nodes are shown in Figure 1b.

## 4 The Control-Flow Perspective in UML 2.0 ADs

In this section we examine the control-flow perspective of UML 2.0 ADs and their ability to represent a series of twenty common control-flow modelling requirements that occur when defining process models. These requirements are described in terms of the Workflow Control Patterns (van der Aalst, ter Hofstede, Kiepuszewski & Barros 2003). The material in this section summarises the findings in (Wohed, van der Aalst, Dumas, ter Hofstede & Russell 2005) thus providing the basis for a comprehensive evaluation of UML 2.0 ADs from multiple perspectives.

### 4.1 Basic control patterns

Basic control-flow patterns define elementary aspects of process control. These are analogous to the definitions of elementary control-flow concepts laid down by the Workflow Management Coalition (WFMC 1999). There are five of these patterns:

- WCP1: *Sequence* – the ability to depict a sequence of activities;
- WCP2: *Parallel split* – the ability to capture a split in a single thread of control into multiple threads of control which can execute in parallel;
- WCP3: *Synchronisation* – the ability to capture a convergence of multiple parallel subprocesses/activities into a single thread of control thus synchronising multiple threads;
- WCP4: *Exclusive choice* – the ability to represent a decision point in a workflow process where one of several branches is chosen; and
- WCP5: *Simple merge* – the ability to depict a point in the workflow process where two or more alternative branches come together without synchronisation.

All five of these patterns can be captured by UML 2.0 ADs. The specific representation of each of these patterns is illustrated in Figure 2.

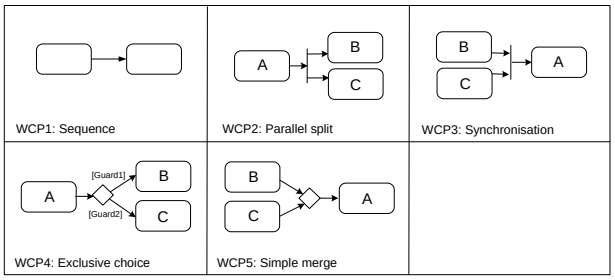


Figure 2: Basic control patterns in UML 2.0 ADs

### 4.2 Advanced branching & synchronisation patterns

This class of patterns corresponds to advanced branching and synchronisation scenarios that often do not have direct realisations in PAIS but are relatively common in real-life business processes. There are four of these patterns:

- WCP6: *Multiple choice* – the ability to represent a divergence of the thread of control into several parallel branches on a selective basis;
- WCP7: *Synchronising merge* – the ability to depict the synchronised convergence of two or more alternative branches;
- WCP8: *Multiple merge* – the ability to represent the unsynchronised convergence of two or more distinct branches. If more than one branch is active, the activity following the merge is started for every activation of every incoming branch; and
- WCP9: *Discriminator* – the ability to depict the convergence of two or more branches such that the first activation of an incoming branch results in the subsequent activity being triggered and subsequent activations of remaining incoming branches are ignored.

The multiple choice, multiple merge and discriminator patterns can be captured directly in UML 2.0 ADs and they are illustrated in Figure 3. The synchronising merge pattern cannot be directly modelled.

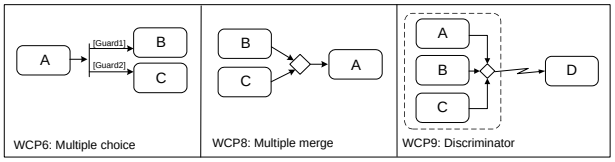


Figure 3: Advanced branching & synchronisation patterns in UML 2.0 ADs

### 4.3 Structural patterns

Structural patterns identify whether the modelling formalism has any restrictions in regard to the way in which processes can be structured (particularly in terms of the type of loops supported and whether a single terminating node is necessary). There are two of these patterns:

- WCP10: *Arbitrary cycles* – the ability to represent loops in a process that have multiple entry or exit points; and
- WCP11: *Implicit termination* – the ability to depict the notion that a given subprocess should be terminated when there are no remaining activities to be completed (i.e. no explicit unique termination node is needed).

Both of these patterns are directly supported in UML 2.0 ADs.

### 4.4 Multiple instance patterns

This class of patterns relates to situations where there can be more than one instance of an activity active at the same time for the same process instance. There are four of these patterns:

- WCP12: *MI without synchronisation* – the ability to initiate multiple instances of an activity within a given process instance;
- WCP13: *MI with a priori design time knowledge* – the ability to initiate multiple instances of an activity within a given process instance. The number of instances is known at design time. Once all instances have completed, a subsequent activity is initiated;
- WCP14: *MI with a priori runtime knowledge* – the ability to initiate multiple instances of an activity within a given process instance. The number of instances varies but is known at runtime before the instances must be created. Once all instances have completed, a subsequent activity is initiated; and
- WCP15: *MI without a priori runtime knowledge* – the ability to initiate multiple instances of an activity within a given process instance. The number of instances varies but is not known at design time or at runtime before the instances must be created. Once all instances have completed, a subsequent activity is initiated. New instances can be created even while other instances are executing or have already completed.

The first three of these patterns can be captured in UML 2,0 ADs as illustrated in Figure 4. The MI without a priori runtime knowledge pattern is not directly supported in UML 2.0 ADs.

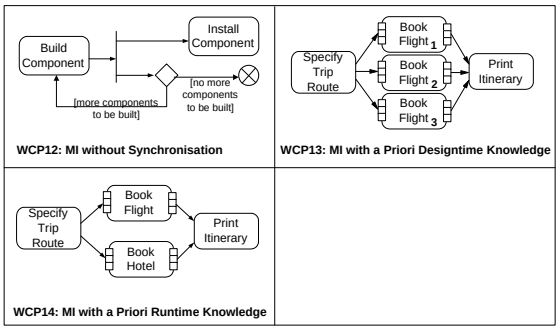


Figure 4: Multiple instance patterns in UML 2.0 ADs

### 4.5 State-based patterns

This class of patterns characterise scenarios in a process where subsequent execution is determined by the state of the process instance. There are three such patterns:

- WCP16: *Deferred choice* – the ability to depict a divergence point in a process where one of several possible branches should be activated. The actual decision on which branch is activated is made by the environment and is deferred to the latest possible moment;
- WCP17: *Interleaved parallel routing* – the ability to depict a set of activities that can be executed in arbitrary order; and
- WCP18: *Milestone* – the ability to depict that a specified activity cannot be commenced until some nominated state is reached which has not expired yet.

Owing to the absence of the notion of state, only the deferred choice pattern can be captured in UML 2.0 ADs. This is illustrated in Figure 5.

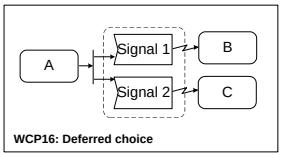


Figure 5: Deferred choice pattern in UML 2.0 ADs

### 4.6 Cancellation patterns

Cancellation patterns characterise the ability of the modelling formalism to represent the potential termination of activities and process instances in certain (specified) circumstances. There are two such patterns:

- WCP19: *Cancel activity* – the ability to depict that an enabled activity should be disabled in some nominated circumstance; and
- WCP20: *Cancel case* – the ability to represent the cancellation of an entire process instance (i.e. all activities relating to the process instance) in some nominated circumstance.

Both of these patterns can be captured in UML 2.0 ADs. The first is illustrated in Figure 6, the second is captured via the ActivityFinalNode construct.

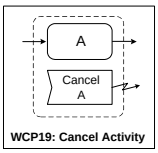


Figure 6: Cancel activity pattern in UML 2.0 ADs

## 5 The Data Perspective in UML 2.0 ADs

Extensions (Russell, ter Hofstede, Edmond & van der Aalst 2005) to the *Workflow Patterns Initiative* have focused on identifying and defining generic constructs that occur in the data perspective of PAIS. In total forty data patterns have been delineated in four distinct groups – data visibility, data interaction, data transfer and data-based routing. In this section, an analysis of UML 2.0 ADs is presented using the data patterns described in (Russell, ter Hofstede, Edmond & van der Aalst 2005).

### 5.1 Data visibility patterns

Data visibility patterns seek to characterise the various ways in which data elements can be defined and utilised within the context of a process. In general, this is determined by the main construct to which the data element is bound as it implies a particular scope in which the data element is visible and capable of being utilised. There are eight patterns which relate to data visibility:

- WDP1: *Task data* – data elements defined and accessible in the context of individual execution instances of a task or activity;

| Nr | Pattern | 2.0 | Nr | Pattern | 2.0 |
|---|---|---|---|---|---|
| | Basic Control | | | Multiple Instance | |
| 1 | Sequence | + | 12 | MI without Synchronization | + |
| 2 | Parallel Split | + | 13 | MI with a priori Design Time Knowledge | + |
| 3 | Synchronisation | + | 14 | MI with a priori Runtime Knowledge | + |
| 4 | Exclusive Choice | + | 15 | MI without a priori Runtime Knowledge | – |
| 5 | Simple Merge | + | | State-based | |
| | Adv. Branching & Synchronisation | | 16 | Deferred Choice | + |
| 6 | Multiple Choice | + | 17 | Interleaved Parallel Routing | – |
| 7 | Synchronising Merge | – | 18 | Milestone | – |
| 8 | Multiple Merge | + | | Cancellation | |
| 9 | Discriminator | + | 19 | Cancel Activity | + |
| | Structural | | 20 | Cancel Case | + |
| 10 | Arbitrary Cycles | + | | | |
| 11 | Implicit Termination | + | | | |

Table 1: Support for Control-Flow Patterns in UML 2.0 ADs

- WDP2: *Block data* – data elements defined by block tasks (i.e. tasks which can be described in terms of a corresponding decomposition) and accessible to the block task and all corresponding components within the associated decomposition;
- WDP3: *Scope data* – data elements bound to a subset of the tasks in a process instance;
- WDP4: *Multiple instance data* – data elements specific to a single execution instance of a task (where the task is able to be executed multiple times);
- WDP5: *Case data* – data elements specific to a process instance which are accessible to all components of the process instance during execution;
- WDP6: *Folder data* – data elements bound to a subset of the tasks in a process definition but accessible to all task instances regardless of the case to which they correspond;
- WDP7: *Workflow data* – data elements accessible to all components in all cases; and
- WDP8: *Environment data* – data elements defined in the operational environment which can be accessed by process elements.

In the case of UML 2.0 ADs, there is support for several of these patterns. The smallest operational unit in the context of these diagrams is the action. This corresponds to the notion of a process element or task and although the notion of task data is not directly supported, there is indirect support in the situation where a local action language is utilised which provides action-specific variables (see (OMG 2004), p.338-9).

Activities serve as the main grouping mechanism in UML 2.0 ADs and they have similar characteristics to the block construct in process definitions. The block data pattern is directly supported through parameters to activities (see (OMG 2004), p.363) which are accessible to all activity components. The concept of attributes (see (OMG 2004), p.341-7) is also provided which allows data elements to be defined which are scoped to a specific activity.

Scope data is not supported. The ActivityGroup construct (see (OMG 2004), p.359) seems to offer something analogous however the semantics of the construct are not defined.

Multiple instance data is directly supported and there are three situations where multiple instances of a given task may arise:

1. Where a task is specifically designated as having multiple instances in the process model – this facility seems to be provided by the ExpansionKind construct (see (OMG 2004), p.394) where the parallel option is chosen forcing parallel execution;
2. Where a task can be triggered multiple times e.g. it is part of a loop. This situation is allowable in UML 2.0 ADs (see (OMG 2004), p.345); and
3. Where two tasks share the same decomposition. This is also supported in UML 2.0 ADs as each activity decomposition is distinct and has a different set of tokens supplied to it at initiation (see (OMG 2004), p.360-1).

Case and folder data are not supported as there does not appear to be the notion of distinct execution instances in UML 2.0 ADs, rather all instances of a process model execute in the same context and are differentiated by distinct sets of control and object tokens flowing through the diagram. Workflow data is directly supported through data objects or ObjectNodes (see (OMG 2004), p.422) which are potentially accessible to all of the components in a UML 2.0 ADs. There does not appear to be the ability within UML 2.0 ADs to refer to data outside of the context of the diagram, and hence environment data is not supported.

### 5.2 Data interaction patterns

Data interaction patterns deal with the various ways in which data elements can be passed between components within a process instance and also with the operating environment (e.g. data transfer between a component of a process and an application, data store or interface that is external to the process). They examine how the characteristics of the individual components can influence the manner in which the trafficking of data elements occurs.

There are six internal data interaction patterns:

- WDP9: *Data elements flowing between task instances*;
- WDP10: *Data elements flowing to a block*;
- WDP11: *Data elements flowing from a block*;
- WDP12: *Data elements flowing to a multiple instance task instance*;
- WDP13: *Data elements flowing from a multiple instance task instance*; and

- WDP14: *Data elements flowing between process instances or cases.*

Data interaction between tasks is directly supported in UML 2.0 ADs by the ObjectNode construct (see (OMG 2004), p.422) which is the standard means of communicating data elements between activities.

Data interaction between block tasks and their decompositions has a similar analogy in UML 2.0 ADs in the form of data passing to and from activities. The standard means of doing this is via parameters (see (OMG 2004), p.363). Both the data interaction block task to sub-workflow and data interaction sub-workflow to block task patterns (WDP10 and WDP11) are directly supported.

Data interaction to and from multiple instance tasks has a direct analogy in UML 2.0 ADs in the ExpansionRegion construct (see (OMG 2004), p.395) which allows nominated regions of a process model to be executed multiple times in parallel (providing the ExpansionKind mode is set to parallel). Data passing into and out of the ExpansionRegion occurs using ExpansionNodes which provide the ability to map distinct sections of the input data set to specific execution instances and similarly completing instances can map their output to a specific section of the output data set. Hence the data interaction to and from multiple instance task patterns (WDP12 and WDP13) are directly supported.

There is no notion of distinct execution cases in UML 2.0 ADs, hence the data interaction – case to case pattern (WDP14) is not supported.

There are 12 external data interaction patterns, characterised by three dimensions:

- The type of process element – task, case or complete process – that is interacting with the environment;
- Whether the interaction is push or pull-based; and
- Whether the interaction is initiated by the process component or the environment.

Difficulties arise when examining UML 2.0 ADs in the context of this class of patterns as the UML approach assumes that an Activity Diagram represents the complete universe of discourse and does not provide the ability to reference or interact with elements that are external to it.

### 5.3 Data transfer patterns

Data transfer patterns focus on the way in which data elements are actually transferred between one process element and another. They aim to capture the various mechanisms by which data elements can be passed across the interface of a process element. There are seven distinct patterns in this category:

- WDP27: *Data transfer by value – incoming* – incoming data elements passed by value;
- WDP28: *Data transfer by value – outgoing* – outgoing data elements passed by value;
- WDP29: *Data transfer – copy in/copy out* – where a process element synchronises data elements with an external data source at commencement and completion;
- WDP30: *Data transfer by reference – without lock* – data elements are communicated between components via a reference to a data element in some mutually accessible location. No concurrency restrictions are implied;
- WDP31: *Data transfer by reference – with lock* – similar to WDP30 except that concurrency restrictions are implied with the receiving component receiving the privilege of read-only or dedicated access to the data element;
- WDP32: *Data transformation – input* – where a transformation function is applied to a data element prior to it being passed to a subsequent component; and
- WDP33: *Data transformation – output* – where a transformation function is applied to a data element prior to it being passed from a previous component.

In the context of UML 2.0 ADs, only three of these patterns are supported: WDP31: data transfer by reference – with lock - is the standard means of passing data elements into an activity as parameters. As UML 2.0 ADs adopt a token-oriented approach to data passing, these parameters – which typically relate to objects – are effectively consumed at activity commencement and only become visible and accessible to other activities once the specific activity to which they were passed has completed and returned them; WDP32: data transformation - input – can be achieved through the ObjectFlow transformation behaviour (see (OMG 2004), p.418) which allows transformation functions to be applied to data tokens as they are passed along connecting edges between activities; and WDP33: data transformation - output – as for pattern WDP32.

### 5.4 Data-based routing patterns

Data-based routing patterns capture the various ways in which data elements can interact with other perspectives and influence the overall execution of the process. There are seven (relatively self-explanatory) patterns in this category:

- WDP34: *Task precondition – data existence*;
- WDP35: *Task precondition – data value*;
- WDP36: *Task postcondition – data existence*;
- WDP37: *Task postcondition – data value*;
- WDP38: *Event-based task trigger*;
- WDP39: *Data-based task trigger*; and
- WDP40: *Data-based routing.*

The majority of these patterns are supported in UML 2.0 ADs. Both action and activity constructs include local preconditions and postconditions based on logical expressions (which may include data elements) framed in OCL (see (OMG 2004), p.336 and p.346). As a consequence, all of the task pre and postcondition patterns (WDP34 - WDP37) are directly supported. The AcceptEventAction construct (see (OMG 2004), p.334) provides direct support for the event-based task triggering pattern. Similarly, there is direct support for data-based routing via the DecisionNode construct and guard conditions on ActivityEdges (see (OMG 2004), p.387 and p.351). The lack of support for persistent state management within UML 2.0 ADs means that the data-based task trigger pattern cannot be captured.

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

Table 2: Support for Data Routing Patterns in UML 2.0 ADs

## 6 The Resource Perspective in UML 2.0 ADs

Recent work (Russell, van der Aalst, ter Hofstede & Edmond. 2005) has focused on the resource perspective and the manner in which work is distributed amongst and managed by the resources associated with a business process. Our investigations have indicated that these patterns are relevant to all forms of PAIS including modelling languages such as XPDL and business process enactment languages such as BPEL4WS. In this section, we examine the resource perspective of UML 2.0 ADs and their expressive power in regard to work distribution.

Forty three workflow resource patterns have been identified in seven distinct groups:

- *Creation patterns* – which correspond to restrictions on the manner in which specific work items can be advertised, allocated and executed by resources;
- *Push patterns* – which describe situations where a PAIS proactively offers or allocates work to resources;
- *Pull patterns* – which characterise scenarios in which resources initiate the identification of work that they are able to undertake and commit to its execution;
- *Detour patterns* – which describe deviations from the normal sequence of state transitions associated with a business process either at the instigation of a resource or the PAIS;
- *Auto-start patterns* – which relate to situations where the execution of work is triggered by specific events or state transitions in the business process;
- *Visibility patterns* – which describe the ability of resources to view the status of work within the PAIS; and
- *Multiple resource patterns* – which describe scenarios where there is a many-to-many relationship between specific work items and the resources undertaking those work items.

In UML 2.0 ADs, the association of a particular action or set of actions with a specific resource is illustrated through the use of the ActivityPartition construct (see (OMG 2004), p.367). This may take many forms although the "swimlanes" notation is probably the most widely adopted means of presentation, where each lane indicates the resource that will be responsible for executing a specific subset (i.e. a partition) of the actions within an activity. Each partition has a name that corresponds to a specific resource or a group of resources to which the contained actions should be allocated at run-time. Partitions may be specified in four distinct ways: Classifier, Instance, Part, and Attribute and Value. The first two of these schemes (i.e. Classifier and Instance) are relevant in the context of resource allocation.

The direct allocation pattern (WRP1) is directly supported in UML 2.0 ADs as the ability to base a partition on a specific instance allows the actions to be associated with a single specified resource. There is no direct notion of roles within UML 2.0 ADs, although where a partition is based on a classifier, it is possible that the contained actions are allocated to multiple objects corresponding to the classifier (i.e. multiple resources). This is analogous to the notion of group allocation within a traditional workflow system. It is up to the individual resources to determine whether one or all of them will execute the assigned actions (see (OMG 2004), p.368-70). Consequently the requirements of the role-based allocation pattern (WRP2) are fully met and the pattern is directly supported. It is not necessary that actions within an activity belong to a partition and have a corresponding resource association, therefore the automatic execution pattern (WRP11) is also directly supported.

None of the other Creation Patterns are supported within UML 2.0 ADs. In the main, this is a consequence of the restrictive manner in which partitions are specified and the lack of support for relationships between distinct partitions. The attribute and value partition specifier seems to offer a means of implementing the deferred allocation pattern (WRP3) by delaying the need to identify a potential resource until run-time, however it is not possible to specify alter-

nate (i.e. parallel) courses of action based on different values of an attribute and even if it were, the necessity to enumerate a distinct course of action for each value (i.e. each potential resource) would make this approach unwieldy. Lack of an integrated authorisation framework, organisational model or access to an execution history also rules out any form of support for the authorisation (WRP4), organisational allocation (WRP9) and history-based allocation (WRP8) patterns respectively.

The execution semantics of a UML 2.0 AD are based on Petri Net token flow, hence actions become "live" once they receive a control-flow token. The resource associated with a given partition can have multiple actions executing (possibly in different partitions) at the same time. There is no notion of scheduling work execution or of resources selecting the work (i.e. the actions) they wish to undertake, hence there is minimal support for the Push, Auto-start or Multiple Resource patterns within UML 2.0 ADs. The following patterns from these classes are directly supported:

- WRP14: *Distribution by allocation - single resource* – the resource(s) associated with a given partition is immediately allocated an action once it is triggered;
- WRP19: *Distribution on enablement* – all actions in a partition are associated with the resource responsible for the partition when they are triggered;
- WRP36: *Commencement on creation* – an action is assumed to be live as soon as it receives a control-flow token;
- WRP39: *Chained execution* – once an action is completed, subsequent actions receive a control-flow token and are triggered immediately; and
- WRP42: *Simultaneous execution* – there are no constraints on how many partitions a given resource can be specified for or how many instances of these can be active at any one time.

None of the Pull, Detour or Visibility patterns are supported.

## 7 Conclusions

The pattern evaluations described in this paper indicate that whilst UML 2.0 ADs have merit as a notation for business process modelling, they are not suitable for all aspects of this type of modelling. They offer comprehensive support for the control-flow and data perspectives allowing the majority of the constructs encountered when analysing these perspectives to be directly captured. However, their suitability for modelling resource-related or organisational aspects of business processes is extremely limited. It is interesting to note that they are not able to capture many of the natural constructs encountered in business processes such as cases and the notion of interaction with the operational environment in which the process functions. These are limitations that they share with most other business process modelling formalisms and reflect the overwhelming emphasis that has been placed on the control-flow and data perspectives in contemporary modelling notations.

The level of support observed for control-flow patterns (see Table 1 for a complete summary[3]) illustrates that there is relatively broad support for capturing the various types of control-flow constructs that may arise in actual business processes. In terms of addressing the patterns that are not directly supported, we would like to make the following recommendations:

- Given the difficulties in capturing state-based patterns, most notably the interleaved parallel routing pattern and the milestone pattern, it may be worthwhile providing direct support for the notion of the place as it exists in Petri nets. Petri net places capture the notion of "waiting state" in a much less restrictive way than the AcceptEventAction construct does;
- UML 2.0 ADs currently do not support the creation of new instances of an activity while other instances of that activity are already running. This could be resolved through extensions to the ExpansionRegion construct to allow further instances to be dynamically created after the activity has started; and
- Given the lack of support for the synchronising merge, a concept similar to the OR-join could be added to UML 2.0 ADs.

The data patterns evaluation is summarised in Table 2. This shows that the data perspective is also well supported. Furthermore, the following remarks can be made:

- There is no notion of cases or distinct process instances in UML 2.0 ADs, hence all data is effectively block-scoped by default and parallel threads of execution occur in the same data space. This could lead to some problematic situations when modelling highly data intensive and/or highly concurrent processes;
- The use of "tokens" as the fundamental underpinning for control and data flow introduces some subtle variations that do not exist in other PAIS (except those based on Petri-nets) – in particular data elements are truly consumed (and cease to exist) when they are passed to an activity for the duration of the activity. This also makes it difficult to actually share a data element/object between concurrent activities. On the other hand, it minimises concurrency problems;
- The token approach provides an effective basis for internal data interaction (and hence all patterns are "+"). In particular, multiple instance data handling seems to be supported for all three multiple instance situations: designated multiple instance tasks, multiply triggered tasks (loops) and block tasks with a common decomposition; and
- There does not seem to be any ability to model things "outside of the model" i.e. in the external environment. Hence there is no real ability to support external data interaction patterns. This may be addressed by using UML ADs in conjunction with other diagrams such as UML interaction, overview and sequence diagrams, but this requires that the relationships between these diagrams be carefully established.

The resource patterns evaluation is summarised in Table 3. As discussed, it indicates that the support in UML 2.0 ADs for the modelling of work distribution directives is relatively minimal. This reinforces the fact that UML 2.0 ADs tend to be control-flow and data-centric and mainly aim to capture simple static routing directives associated with actions. They do not provide a means of representing the subtleties associated with selective work allocation across a range

[3] A "+" in the table indicates *direct support* for the pattern (i.e. there is a construct in the language that directly supports the pattern).

| Nr | Pattern | | Nr | Pattern | |
|---|---|---|---|---|---|
| | Creation Patterns | | | Pull Patterns (cont.) | |
| 1 | Direct Allocation | + | 24 | System-Determ. Wk Queue Cont. | – |
| 2 | Role-Based Allocation | + | 25 | Resource-Determ. Wk Queue Cont. | – |
| 3 | Deferred Allocation | – | 26 | Selection Autonomy | – |
| 4 | Authorisation | – | | Detour Patterns | |
| 5 | Separation of Duties | – | 27 | Delegation | – |
| 6 | Case Handling | – | 28 | Escalation | – |
| 7 | Retain Familiar | – | 29 | Deallocation | – |
| 8 | Capability-Based Allocation | – | 30 | Stateful Reallocation | – |
| 9 | History-Based Allocation | – | 31 | Stateless Reallocation | – |
| 10 | Organisational Allocation | – | 32 | Suspension/Resumption | – |
| 11 | Automatic Execution | + | 33 | Skip | – |
| | Push Patterns | | 34 | Redo | – |
| 12 | Distrib. by Offer - Single Resource | – | 35 | Pre-Do | – |
| 13 | Distrib. by Offer - Multiple Resources | – | | Auto-Start Patterns | |
| 14 | Distrib. by Allocation - Single Resource | + | 36 | Commencement on Creation | + |
| 15 | Random Allocation | – | 37 | Creation on Allocation | – |
| 16 | Round Robin Allocation | – | 38 | Piled Execution | – |
| 17 | Shortest Queue | – | 39 | Chained Execution | + |
| 18 | Early Distribution | – | | Visibility Patterns | |
| 19 | Distribution on Enablement | + | 40 | Conf. Unalloc. Work Item Visibility | – |
| 20 | Late Distribution | – | 41 | Conf. Alloc. Work Item Visibility | – |
| | Pull Patterns | | | Multiple Resource Patterns | |
| 21 | Resource-Init. Allocation | – | 42 | Simultaneous Execution | + |
| 22 | Resource-Init. Exec. - Alloc. Wk Items | – | 43 | Additional Resource | – |
| 23 | Resource-Init. Exec. - Offer. Wk Items | – | | | |

Table 3: Support for Resource Patterns in UML 2.0 ADs

of possible resources and the management of those work items at run-time. In particular, there is no real support for modelling any form of work distribution other than direct allocation or role-based allocation. There is no opportunity to utilise data resources (either within the model or externally from the environment) thus any opportunity for modelling organisational, history-based or capability-based allocation is obviated. Similarly, there is no support for specifying any form of work distribution algorithm or employing varying styles of work distribution (e.g. push vs pull, offer vs allocation).

Other observations arising from the resource patterns analysis include:

- The fact that the partitions can result in actions being simultaneously allocated to more than one resource can lead to difficulties where a means of providing role-based work allocation to a single resource is required. It is important to note that the resolution of this situation must be addressed as part of the implementation of the actions within the Activity Diagram; and
- The ability to use OCL statements in the specification of partitions (and also for specifying relationships between partitions) would enhance the capability of UML 2.0 ADs to capture possible resource allocations, both in terms of precision and the range of work allocation strategies that could be represented.

## References


Bubenko, J., Persson, A. & Stirna, J. (2001), EKD user guide, Technical report, Royal Institute of Technology (KTH) and Stockholm University, Stockholm, Sweden.

Curtis, B., Kellner, M. & Over, J. (1992), 'Process modelling', *Communications of the ACM* **35**(9), 75–90.

Dumas, M. & ter Hofstede, A. (2001), UML activity diagrams as a workflow specification language, *in* M. Gogolla & C. Kobryn, eds, 'Proceedings of the Fourth International Conference on the Unified Modeling Language (UML 2001)', LNCS 2185, Springer, Toronto, Canada, pp. 76–90.

Eriksson, H. & Penker, M. (2000), *Business Modeling with UML*, OMG Press, New York.

Humphrey, W. & Feiler, P. H. (1992), Software process development and enactment : Concepts and definitions, Technical Report SEI-92-TR-4, SEI Institute, Pittsburgh, USA.

Jablonski, S. & Bussler, C. (1996), *Workflow Management: Modeling Concepts, Architecture and Implementation*, Thomson Computer Press, London, UK.

Kueng, P., Kawalek, P. & Bichler, P. (1996), How to compose an object-oriented business process model?, *in* S. Brinkkemper, K. Lyytinen & R. Welke, eds, 'Proceedings of the IFIP WG8.1/WG8.2 Working Conference', Atlanta, GA, USA.

Marshall, C. (1999), *Enterprise Modeling with UML*, Addison Wesley, Reading.

Mendling, J., Neumann, G. & Nüttgens, M. (2005), A comparison of XML interchange formats for business process modelling, *in* L. Fischer, ed., 'Workflow Handbook 2005', Workflow Management Coalition, Lighthouse Point, Florida, USA, pp. 185–198.

OMG (2004), UML 2.0 superstructure specification, Technical report. `http://www.omg.org/cgi-bin/doc?ptc/2004-10-02`.

Opdahl, A. & Henderson-Sellers, B. (2002), 'Ontological evaluation of the UML using the Bunge-Wand-Weber model', *Software and System Modeling* **1**(1), 43–67.

Rolland, C. (1997), A primer for method engineering, *in* 'Proceedings of the INFormatique des ORganisations et Systèmes d'Information et de Décision (INFORSID'97)', Toulouse, France.

Rolland, C. (1998), A comprehensive view of process engineering, *in* B. Pernici & C. Thanos, eds, 'Proceedings of the 10th International Conference on Advanced Information Systems Engineering (CAiSE'98)', Vol. 1413 of *Lecture Notes in Computer Science*, Springer, Pisa, Italy.

Russell, N., ter Hofstede, A., Edmond, D. & van der Aalst, W. (2005), Workflow data patterns: Identification, representation and tool support, *in* 'Proceedings of the 25th International Conference on Conceptual Modeling (ER'2005)', Springer, Klagenfurt, Austria.

Russell, N., van der Aalst, W., ter Hofstede, A. & Edmond., D. (2005), Workflow resource patterns: Identification, representation and tool support., *in* O. Pastor & J. Falcao é Cunha, eds, 'Proceedings of the 17th Conference on Advanced Information Systems Engineering (CAiSE'05)', Vol. 3520 of *Lecture Notes in Computer Science*, Springer, Porto, Portugal, pp. 216–232.

Scheer, A.-W. (2000), *ARIS - Business Process Modelling*, Springer, Berlin, Germany.

van der Aalst, W., ter Hofstede, A., Kiepuszewski, B. & Barros, A. (2003), 'Workflow patterns', *Distributed and Parallel Databases* **14**(3), 5–51.

Vernadat, F. (1996), *Enterprise Modeling and Integration*, Chapman and Hall, London.

WFMC (1999), Workflow management coalition terminology and glossary, document status - issue 3.0, Technical Report WFMC-TC-1011, Workflow Management Coalition. `http://www.wfmc.org/standards/docs/TC-1011_term_glossary_v3.pdf`.

White, S. (2004), Process modeling notations and workflow patterns, *in* L. Fischer, ed., 'Workflow Handbook 2004', Future Strategies Inc., Lighthouse Point, FL, USA., pp. 265–294.

Wohed, P., Perjons, E., Dumas, M. & ter Hofstede, A. (2003), Pattern based analysis of EAI languages - the case of the business modeling language, *in* O. Camp & M. Piattini, eds, 'Proceedings of the 5th International Conference on Enterprise Information Systems (ICEIS 2003)', Vol. 3, Escola Superior de Tecnologia do Instituto Politecnico de Setubal, Angers, France, pp. 174–184.

Wohed, P., van der Aalst, W., Dumas, M., ter Hofstede, A. & Russell, N. (2005), Pattern-based analysis of UML activity diagrams, *in* 'Proceedings of the 25th International Conference on Conceptual Modeling (ER'2005)', Springer, Klagenfurt, Austria.

zur Muehlen, M. & Rosemann, M. (2004), Multi-paradigm process management, *in* J. Grundspenkis & M. Kirikova, eds, 'Proceedings of the Fifth Workshop on Business Process Modeling, Development, and Support (BPMDS '04), held in conjunction with the Conference on Advanced Information Systems Engineering (CAiSE) 2004', Vol. 2, Faculty of Computer Science and Information Technology, Riga Technical University, Riga, Latvia, pp. 169–175.