Valmont: a Language for Workflow Programming *

Daniel K.C. Chan[†]

INRIA - Rocquencourt
Domaine de Voluceau, B.P. 105
78153 Le Chesnay Cedex, France
Daniel.Chan@inria.fr

Karl R.P.H. Leung

Department of Computing
Hong Kong Polytechnic University
Hunghom, Hong Kong
cskleung@comp.polyu.edu.hk

Abstract

The workflow paradigm is advancing the scope of process modelling by supporting complex control flows, rich process structures, and the integration of heterogeneous and legacy systems. However, studies of the paradigm have been seriously hindered by the lack of a concrete framework. In order to improve understanding of the paradigm, enable studying of interaction between different features, and facilitate the development of more advanced features, this paper presents a framework in the form of a workflow language called Valmont. Unlike many other proposals, Valmont captures all the fundamental elements of the workflow paradigm at an abstraction level that is suitable for workflow application designers. More specifically, it provides a rich organisation model that can capture distribution using domains, an information model that also records presentation details, as well as a sophisticated process model that supports active rules and workflow transactions. This paper presents an overview of Valmont by providing a solution to the ISPW-6 Software Process Example. It also serves to provide an insight into the needs of novel application domains some of which remain to be addressed by the next generation of workflow management systems.

†Supported by a Marie Curie Fellowship from the European Commission (ERBFMBICT960653).

1 Introduction

Many traditional process models are geared towards capturing the operational details of one function or a small number of functions within a business enterprise. Such process models are usually rather simple and hence do not reflect the processing reality of an enterprise. Besides, there is also the difficulty of dealing with different functions modelled using various approaches and implemented upon diverse platforms. More often than not, imposing a uniform approach and platform over all functions is not a practical and realistic option. An effective process model should therefore address the need of integrating applications developed using mixed approaches and platforms. The workflow paradigm aims to address these issues by providing workflow models and cooperative software environments within which different aspects of a business enterprise can be modelled, co-ordinated, and monitored. Workflow systems have also been acknowledged as an important enabling technology for business process re-engineering [24] which aims to transform existing business functions into more responsive and flexible ones.

To cope with the complexity of the target applications, workflow models support complex control flows and rich process structures. Control flows are no longer limited to linear flows. Complex conditions can be imposed on individual flows, collections of flows, as well as the start and termination of processes. Moreover, control flows can be influenced by external actions that may be initiated by the user or the workflow system. For instance, temporal events which occur at specified time or periodically can be specified. Processes can run concurrently and may even have sub-processes that can interact with one another. Besides, human resources and legacy systems can be captured and integrated into workflow models. In other words, workflow models provide a comprehensive approach to business process modelling.

The success enjoys by the workflow paradigm in business process re-engineering is attracting a lot of attention from other application domains such as scientific and engineering applications [21]. However, the long list of fundamental workflow features [23] needs to be extended to meet the requirements of the new application domains. Studies and discussions of such extensions have been hampered by the lack of a concrete framework. Both the model and semantics of many workflow systems have not been addressed properly. The aim of this paper is to introduce a concrete framework through the presentation of a workflow language called *Valmont*. It differs from other workflow languages and proposals in that it captures all the fundamental elements of the workflow paradigm (namely organisation model, information model, process model, and their relationships) at an abstraction level that is suitable for the user rather than machine manipulation.

It should be noted that *Valmont* is only a textual process description for a workflow system that is primarily operated in an interactive graphical environment. Having a textual representation is important for a number of reasons. If a workflow system is to be around for a considerable length of time, it may well require upgrading every now and then. Given the rapid development in workflow products, upgrading may be realised by switching to a different implementation platform or workflow engine. A workflow language can ease the transition in both circumstances by providing a bridge between different systems. Besides being part of a workflow system, it can also be used for process documentation. It should be noted that these advantages hold even when many workflow design steps are actually performed interactively using forms and graphical tools.

The rest of the paper is structured as follows. Section 2 discusses related work on process models, workflow models, and workflow languages. Section 3 introduces the requirements of the novel application domains. Section 4 presents *Valmont* using the ISPW-6 Software Process Example. Section 5 briefly describes the architecture of the system supporting *Valmont*. Section 6 concludes.

2 Related Work

Traditional techniques such as data flow diagrams are found to be unsuitable for modelling complex processes due to their inability to capture the dynamic aspects of such processes. Petri nets, which are often used to describe and analyse concurrent systems [4], address the inadequacy concerning modelling process dynamics. High-level Petri nets [15] such as predicate/transition nets and colored Petri nets raise the abstraction level closer to that of user applications and hence become more useful for process modelling. An attempt to model workflows using Petri nets is given in [11]. Statecharts [13], which are designed for modelling reactive systems, allow concurrency and process abstraction to be expressed. A far less formal approach to process modelling based on the speech act theory has attracted a lot of attention [19]. A simplified form of it containing four phases: *request*, *commitment*, *performance*, and *evaluation* has been deployed as the modelling framework for Action Workflow [18]. Nevertheless, some features of the workflow paradigm such as the organisation model have not been captured by the above-mentioned proposals.

Recently, the Workflow Management Coalition has issued a proposal for a workflow language [22]. The proposal and *Valmont* have close similarities but there are also significant differences. The proposal sketches the components of a language that is supposed to be supported by all workflow systems so as to facilitate interoperability of workflow systems and translation between different representations of workflow models. It focuses mainly on the attributes of entities which are only cursorily considered in this paper. However, no concrete syntax is provided and a number of fundamental features are not addressed at all. *Valmont* is more complete in both coverage and syntax. However, it would be more constructive to look at the two languages as complementary to one another.

It is argued in [12] that C&Co, a C-based general purpose programming language, is sufficient to serve as a workflow language. Distributed concurrent processes can be created and communication between such processes relies on the use of write-once variables. Transaction primitives supporting compensation and the two-phase commit protocol are provided. The ability of C&Co to express workflow computation and control is beyond dispute but the level of abstraction provided is questionable. Also no organisation model and user interaction are taken into account. Perhaps, C&Co should be seen as an implementation language for the realisation of workflow systems. Like C&Co, WADL [10] captures only the control flows and transaction properties of tasks. Unlike C&Co, it provides a higher level of abstraction supporting sequential, parallel, alternative, and non-deterministic flows. The last two are not directly supported in *Valmont* but can be modelled easily. The workflow language of the METEOR system [17] (consisting of WFSL and TSL) comes much closer to be a "real" workflow language with the support of simple control flows, complex dependencies on data flows, process abstraction, transaction support and exception handling; however, several elements, such as the organisation model, are still missing

The workflow language of FlowMark from IBM [14] bears a close resemblance to *Valmont* but lacks a number of advanced features that are supported by *Valmont*. These features include richer and more declarative activity assignment constraints, active rules providing a more general framework for handling exceptions, and high-level transactions [1] that are absent in FlowMark.

Among the fundamental elements of the workflow paradigm, the organisation model is probably the least studied element. The only work that the authors are aware of is reported in [5] which advocates an organisation model containing three components: organisation structure, organisation expression, and activity assignment policy. All three components show strong influences from the relational database model. In an organisation structure one “relation” is used for every agent type and every relationship between agent types. An organisation expression is essentially a “view”. An activity assignment policy specifies the domain - a conceptual geographical location represented by a domain identifier - of a workflow activity as well as the organisation expression that returns a set of eligible agents for the activity. Unlike *Valmont*, only activities can be given a domain. Furthermore, the relational query language used in organisation expressions may not be computationally powerful enough, for instance it is not clear if recursive search for superior is possible. Generally speaking and evidenced by all examples shown in [5], agent relationships are used only for traversing between agents and there is no need for them to be “first-class” entities in the model.

3 Workflow Language Requirements

It is a general consensus that the power of the workflow paradigm lies in its ability to integrate. The ultimate goal is to have the ability to exchange data and control information over a number of independent heterogeneous systems. It is very likely that these systems are installed in different sites distributed over a large geographical area. Site identification and the allocation of processes to sites are therefore important.

Besides the requirements arising from its operational environment, the workflow paradigm is supposed to provide powerful modelling capability. It is expected to support rich control dependency including temporal behaviour and event-based behaviour. The latter also includes failure and exception handling. On the data side, it should provide security measures to ensure their integrity. Concurrent access to shared data should be carefully controlled. Data should be protected from application and system failures.

Like many large applications, it should be possible to develop a workflow system in an incremental fashion. This can be translated to the need of some abstraction mechanism which allows definitions to be reused. It is also identified that dynamic modification is a very common practice in many application domains. Moreover, management information should be collected during execution so that monitoring can be performed during execution and review can be carried out afterwards. Discussion about the requirements to support the software process can be found in [7].

4 Overview of Valmont

4.1 The ISPW-6 Software Process Example

This section presents *Valmont* using a well-known software process example commonly called the ISPW-6 Software Process Example [16]. It is the first benchmark application example to facilitate objective assessment and comparison between software process models. The example concerns the execution and management of a change of requirements for an existing software component. The process is initiated in response to the modification of a requirement unit. It begins with a project leader scheduling and assigning the various engineering tasks in the process. These include modification of the corresponding design unit, modification of the test plan, modification of the unit test, and testing of the modified unit. During the execution of these activities the project leader monitors their progress. The process description specifies constraints on the process including the ordering of some of the tasks, the kinds of personnel responsible for each task, as well as conditions for task initiation and termination. The tasks and the (simplified) control flows between them are depicted in Figure 1. “Monitor Progress” is a task that interacts with all the other tasks in “Develop Change and Test Unit”.

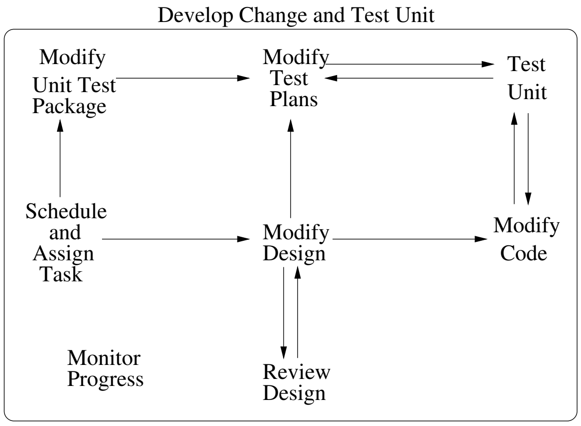

Figure 1. Control Flows in the ISPW-6 Software Change Process

The *Valmont* specifications given in the following subsections represent only a partial solution to the ISPW-6 Example. The emphasis is on the delicate co-ordination that can be expressed in *Valmont*. Moreover, some features are expressed only once in the specifications as opposed to their multiple uses in the ISPW-6 Example. Further discussion of the individual tasks are given when their corresponding specifications are examined.

4.2 The Liaison Workflow Model

The *Liaison* workflow model, on which *Valmont* is based, is derived from the WIDE workflow model [6]. It is an extension of the reference workflow model proposed by the Workflow Management Coalition [23]. Like the reference workflow model, it captures the fundamental elements of the workflow paradigm: organisation model, information model, process model, and their relationships. Unlike the reference workflow model, it supports a rich organisation model, sophisticated activity assignment constraints, dynamic control flows including the use of active rules, complex process structures, and workflow transactions. *Liaison* extends the WIDE model with more elaborate process event dependencies, dynamic process evolution support, and process integration support. *Valmont*, which is a workflow language for *Liaison*. It is based on a workflow language defined earlier by the first author [8].

4.2.1 Organisation Model

The organisation model registers primarily the organisation structure and resources. It records information about individual employees (*staff*), functional positions held by staff (*position*), groups of staff (*team*) that are put together to serve some transaction, as well as non-human resources (*tool*) such as machines and software. A staff member can hold several positions as well as participate in a number of teams possibly in different capacities according to the positions held. Staff, positions, teams, and tools are collectively referred to as *agents*. All agents are associated with a *domain* which is usually used to model geographical sites or functional units of an organisation.

The organisation model also captures relationships between staff, between positions, and between teams. These relationships are useful for searching agent replacement. The *deputy* relationship allows one staff member to deputise for another that is not available to carry out an activity. The *position hierarchy* captures the *accountability* relationship between positions. When a staff member is not available, a task can be performed instead by his or her senior to whom the staff member is accountable for. The *team hierarchy* recording the *inclusion* relationship serves a similar purpose. When a team member is not available, the team leader can take over. However, if no one in the team is available, the leader of the *affiliated* team which includes the original team can be brought in.

4.2.2 Information Model

The information model defines the data used in a workflow process, governs the operations that can be performed on the data, as well as controls the scope and presentation of the data. Data can have one of two possible scopes: *global* and *local*. Global data are shared by all workflow processes. They are persistent and are stored usually in either databases or files. Local data are only accessible from within one workflow process and are shared between activities within the same process. They are managed by the workflow system and are in many ways not unlike global data.

The *entity* data type is introduced to capture external data and their associated operations. Every entity may have a set of associated operations that are often completely different from any other entities. Nevertheless, for the purpose of workflow applications, it is sufficient to treat all entities to be of the same type at the expense of having to perform some type checking dynamically at run-time.

The *form* data type is probably the most distinguishing type in the information model. It resembles the record type in programming languages except that it also contains presentation information. The presentation information are important due to the interactive nature of workflow systems - a lot of information are presented and captured using forms displayed on user terminals. Presentation information include headings, field labels, and formatting information. Each field is typed and can be used to show or update data variables. Default field values can also be defined. A form type can be derived from another form type possibly by removing fields. Moreover a derived form can relax or restrict operations that can be performed on a field. This mechanism encourages and supports re-usability which reduces the maintenance effort.

The information model supports the well-known base types: *integer*, *string*, and so forth. It also supports *list* as a container type whose elements can be of any type including list, entity, and form. Actually types are orthogonal and can be combined freely.

4.2.3 Process Model

The use of activity abstraction in the process model allows an application to be defined in an incremental fashion and often the resultant design is more comprehensible. In other words, an activity representing a unit of work at one level can be refined into a network of activities in the next level. Activities at the bottom level, so to speak, are the building blocks of the process where actions are actually carried out. The bottom-level activities are referred to as *base tasks* while other activities are called *tasks*.

Every activity definition can contain the following components: (1) a *pre-condition* that has to be satisfied for the activity to start, (2) actions performed by the activity, (3) a *post-condition* that has to hold for the activity to terminate, (4) a *role constraint* about assigning the activity to an agent, (5) a *schedule* for the execution of the activity, and (6) handlers for system-defined as well as user-defined *exceptions*. The action part of an activity varies depending on whether the activity is a task or a base task. For a task it captures the control flows between activities in the next level. For a base task it captures some computation which may be performed over the information model or some external operation performed by the user. Consequently, role constraints may differ between tasks and base tasks. For a task a role constraint may specify dependency between activity assignments of lower-level activities. For a base task a role constraint is always about the base task alone. Base tasks have the option of specifying the forms that can be used for interaction with the user.

A task can be highly dynamic as its execution can be influenced by exceptions raised by the user through the interactive tools provided (e.g. abort a task) or by the workflow system itself (e.g. a *temporal event* that occurs regularly). Therefore, an activity can be at different *states* at different times. Operations are executed only when an activity is in the *running* state. Control flows between activities do not have to be structured in a linear fashion. An activity can be followed by (or *fork*) a number of activities while multiple activities can precede (or *join*) a single activity.

4.3 Organisation Specification

The organisation model can be used to capture an operational environment having multiple sites. Each site can be considered as a domain. The hierarchy that often exists between sites can be modelled as a hierarchy between domains. In the (partial) organisation specification "Software_Services" given in line 6-27 of Figure 2, the domain qualifier LOCAL (line 6) is actually a domain parameter which is bound to the domain in which the workflow process executes. Parameterised domains allow reuse of organisation models. All components defined in a specification are qualified by the domain of the specification. Domain qualification is however optional as is the case for "Sino_French" defined in line 1-5. *Valmont* allows one specification to import other specifications. The USES clause in line 7 of the "Software_Services" specification allows all components defined in the "Sino_French" specification to be used in the "Software_Services" specification. Often connections to the imported models are to be established, one such connection between "Software_Services" and "Sino_French" is established in line 25 which will be explained later.

Properties of individual employees are captured using the STAFF definition. Attributes appeared after the keyword HAS are user-defined attributes which correspond to the *extended attribute list* defined in [22]. Attribute values can be of any type, for instance "skills" is a list-valued attribute and "degree" is a string-valued attribute. Line 28-38 of Figure 2 show how an organisation model can be populated with instances using the REGISTER statements. Line 30-32 register

```
1  ORGANISATION_MODEL Sino_French
     ...
2    TEAM Configuration_Control_Board
3    ...
4    END_TEAM;
5  END_MODEL

6  ORGANISATION_MODEL Software_Services OF LOCAL
7    USES ORGANISATION_MODEL Sino_French;
     ...
8    STAFF HAS
9      LIST OF STRING: skills;
10     LIST OF STRING: languages;
11     STRING: degree;
12   END_STAFF;

13   POSITION Design_Engineer SUPERIOR Project_Leader
14     ...
15   END_POSITION;

16   POSITION Quality_Assurance_Engineer SUPERIOR Project_Leader
17     ...
18   END_POSITION;

19   POSITION Software_Engineer SUPERIOR Project_Leader
20     ...
21   END_POSITION;

22   POSITION Project_Leader
23     ...
24   END_POSITION;
     ...
25   TEAM Project_Team AFFILIATION Configuration_Control_Board HAS
     ...
26   END_TEAM;
27 END_MODEL

28 REGISTER TEAM Configuration_Control_Board IN Sino_French
29   (LEADER) (Frank);

30 REGISTER STAFF IN Software_Services OF France
31   (SELF, DEPUTY, languages, skills, degree)
32   [ (Andy, [Billy], ["chinese", "english", "french"],
       ["databases", "software engineering"], "Ph.D."), ... ];

33 REGISTER POSITION Design_Engineer IN Software_Services OF France
34   (SELF, SUPERIOR)
35   [ (Andy, Edward), (Billy, Edward), ... ];
   ...
36 REGISTER TEAM Project_Team IN Software_Services OF France
37   (LEADER, MEMBERS)
38   (Edward, [Andy, Bob, Chris, David AS Software_Engineer]);
```

Figure 2. A Partial Organisation Model

an employee identified as "Andy" with four attribute values. The first attribute "DEPUTY" is pre-defined and provides a list of employees who can stand in for the employee being registered.

Positions are defined using the POSITION definition. Apart from pre-defined attributes, user-defined attributes can be introduced as in the STAFF definition. The optional accountability relationship can also be specified. In line 16-18, the position "Quality_Assurance_Engineer" is defined to be accountable to "Project_Leader" but no such relationship is defined for “Project_Leader” in line 22-24. Line 33-35 show the registration of two “Design_Engineer”’s identified as “Andy” and “Billy”. They are both accountable to “Edward”. Each employee is assigned to at least one position but can be more than one (see “David” in line 38).

Organisations are becoming more team-oriented rather than function-oriented as the former helps to eliminate unnecessary control resulting in better responsiveness and flexibility. Teams can be defined using the TEAM definition, for example in line 2-4 and 25-26. The pre-defined attribute “LEADER” of “Project_Team” is registered in line 38. Since a team member can hold more than one position, it is sometime necessary to specify the capacity of a team member. In line 38, “David” serves in the “Project_Team” as a “Software_Engineer” rather than other capacities (e.g. “Quality_Assurance_Engineer”) that he may hold. Teams can form a team hierarchy using the inclusion relationship. For instance, the “Project_Team” in “France” is included in the “Configuration_Control_Board” team. This is specified using the AFFILIATION clause in line 25.

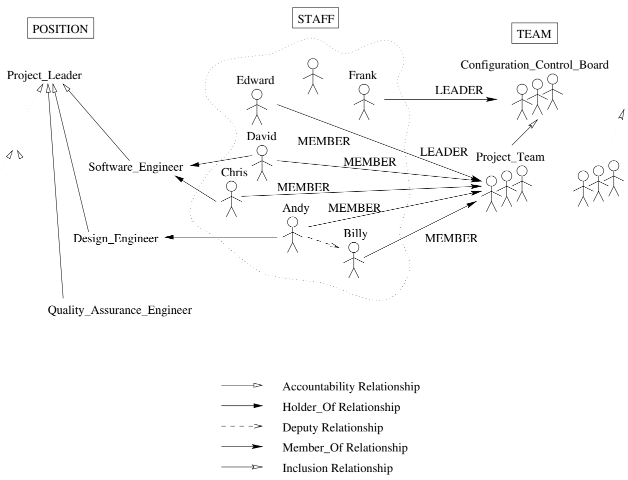

Figure 3. Relationships between Agents

The relationships between agents are summarised using different kinds of arrows in Figure 3.

4.4 Information Specification

The information model is actually made up of two models - the *form model* and the *data model*. The form model contains representation details of forms that can be used to interact with the user. The data model can be used to define global and local data. The next two subsections provide partial specifications of the form and data models used in the solution.

```
1   FORM_MODEL Software_Services OF France
2     FORM Design_Change_Form
3       HEADING "DESIGN CHANGE FORM";
4       SPACE;
5       LABEL "Software Design" LINKS ENTITY: design;
6       NEW_LINE; SPACE;
7       LABEL "New Requirements" LINKS TEXT: requirements_change;
8       NEW_LINE; SPACE;
9       LABEL "Additional Recommendation" LINKS TEXT: fix_design_feedback;
10    END_FORM;

11    DERIVED_FORM First_Design_Change_Form
        DERIVED_FROM Design_Change_Form
12      WITHOUT fix_design_feedback;
13      ONLY READ FOR requirements_change;
14    END_FORM;

15    DERIVED_FORM More_Design_Change_Form
        DERIVED_FROM Design_Change_Form
16      ONLY READ FOR requirements_change, fix_design_feedback;
17    END_FORM;
      ...
18  END_MODEL
```

Figure 4. A Partial Form Model

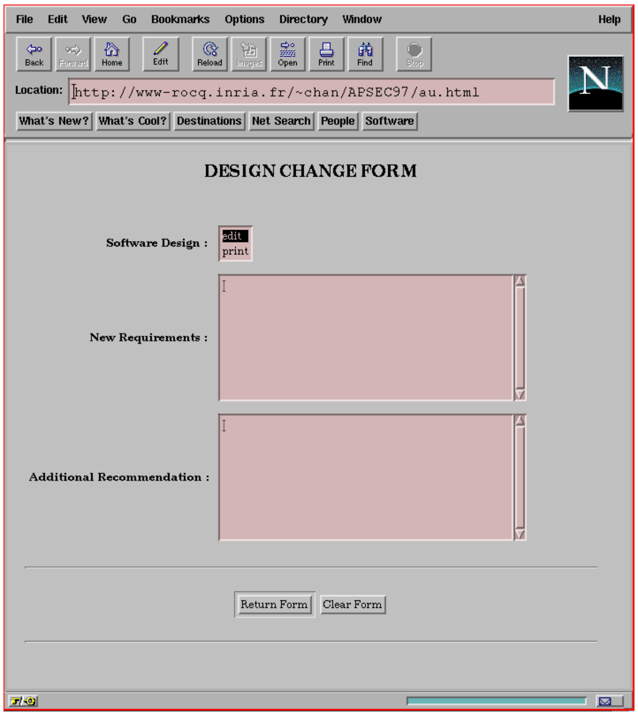

Figure 5. Design Change Form

4.4.1 Form Specification

A form specification contains representation details concerning the display of a form and the types of the data that can appear in a form. The “Design_Change_Form” is defined in line 2-10 in Figure 4 and the appearance of the form is given in Figure 5. The heading statement (line 3) prints a string on the screen while NEW_LINE and SPACE correspond to starting a new line and producing some vertical spacing. Each LABEL statement can generate a number of buttons or boxes which may be labelled. For example, line 9 generates a box next to the label “Additional Recommendation” to represent a link to a piece of data of type TEXT. It should be noted that fields in a form are references to local data and they are not value holders themselves. The type information of the fields are included to facilitate type checking and efficient formatting.

The “First_Design_Change_Form” is derived from the “Design_Change_Form” and differs from the latter in two ways. First, the field “fix_design_feedback” is removed (line 12). Second, the field “requirements_change” cannot be updated (line 13). Similarly, “More_Design_Change_Form” is derived from the “Design_Change_Form”. It restricts access to fields “requirements_change” and “fix_design_feedback” to read-only. Derived forms are used to control the scope and access of local data by means of structural changes, access constraints, and initial values for fields before the form is displayed. Form derivation is simply a syntactic sugaring for defining new forms and does not imply any relationship between forms.

4.4.2 Data Specification

```
1  DATA_MODEL Software_Services OF France
2    ENTITY design "~ss/ispw6/design.txt"
3      HAS
4      edit "emacs $design";
5      print "lpr $design";
6    END_ENTITY;

7    ENTITY source_code ...
8      ...
9    END_ENTITY;

10   ENTITY object_code ...
11     ...
12   END_ENTITY;

13   ENTITY test_unit_package ...
14     ...
15   END_ENTITY;

16   VARIABLE
17     STRING: fix_design, test_results;
18     BOOLEAN: fix_code, fix_test;
19     TIME: test_unit_start, test_unit_feeback;
20     TEXT: requirements_change, fix_design_feedback, no_of_defects,
21        fix_code_feedback, fix_test_feedback, test_plans;
22 END_MODEL
```

Figure 6. A Partial Data Model

The data model given in Figure 6 contains two categories of data. Line 2-6 captures an external document as an entity and defines two associated operations: “edit” and “print”. Operation associated with an entity are usually listed in a pull-down menu in a form. It is also possible to copy an entity from one field to another. Line 17-21 defines variables of various types that can be used with forms as described in the next section.

4.5 Process Specification

The process model primarily captures various relationships between processes: the refinement relationship, control flows, and data flows. The ensuing discussion is based on the specifications given in Figure 7 and 8.

```
1  WORKFLOW_MODEL ISPW-6 OF France

2  USES ORGANISATION_MODEL Software_Services;
3       FORM_MODEL Software_Services;
4       DATA_MODEL Software_Services;

5  CONTROL
6    START Schedule_Task;
7    Schedule_Task ENABLES Perform_Change;
8    END Perform_Change;

9  EXCEPTION
10   Test_Unit_Delayed
11     WHEN AT(test_unit_start) AND NOT STARTED(Test_Unit)
12     DOES NOTIFY(Project_Team.LEADER, "...");

13   Test_Unit_Overdue
14     WHEN ELAPSED(Test_Unit) > test_unit_length
15     DOES NOTIFY(Project_Team.LEADER, "...");

16   Review_Design_Completed
17     WHEN COMPLETED(Review_Design)
18     DOES NOTIFY(Project_Team.LEADER, "..." + no_of_defects);


19 TASK Perform_Change

20   START Modify_Design, Modify_Test_Plans;

21   Modify_Design ENABLES Review_Design;
22   Review_Design IF (fix_design IN LIST["minor changes", "major changes"])
23     ENABLES Modify_Design;
24   EACH(STARTED(Modify_Design),
25       Test_Unit IF (fix_code = TRUE))
26     ENABLES Modify_Code;
27   Modify_Design ENABLES COMPLETED(Modify_Code);

28   EACH((Modify_Test_Plans, Modify_Design),
29       Test_Unit IF (fix_test = TRUE))
30     ENABLES Modify_Test_Unit_Package;
31   Modify_Test_Unit_Package, Modify_Code ENABLES Test_Unit;

32   END Test_Unit;

33 END_TASK;
```

Figure 7. Tasks Modelling the ISPW-6 Software Process Example

The workflow process definition begins with importing an organisation model (line 2), a form model (line 3), and a data model (line 4) relevant to the application.

```
34 BASE_TASK Schedule_Task
35   VIEW
36     Schedule_Form(test_unit_start, test_unit_length);
37   PRE_CONDITION
38     Schedule_Form.start IS NULL;
39     Schedule_Form.length IS NULL;
40   MESSAGE
41     "Please fill in the start time and duration.";
42   POST_CONDITION
43     EVERY Schedule_Form IS NOT NULL;
44   ROLE
45     STAFF Project_Team.LEADER;
46 END_TASK;

47 BASE_TASK Modify_Design
48   VIEW
49     First_Design_Change_Form(requirements_change, design)
50         IF fix_design IS NULL;
51     More_Design_Change_Form(requirements_change, design,
           fix_design_feedback)
52         IF fix_design IN LIST["minor changes", "major changes"];
     ...
53   ROLE
54     STAFF Project_Team.design_engineer;
55 END_TASK;

56 BASE_TASK Review_Design
57   VIEW
58     Review_Form(requirements_change, design,
       fix_design_feedback, fix_design, no_of_defects);
     ...
59   ROLE
60     STAFF Project_Team.design_engineer;
61     STAFF Project_Team.quality_assurance_engineer;
62     STAFF 2 Project_Team.software_engineer;
63 END_TASK;

64 BASE_TASK Modify_Code
65   VIEW
66     First_Code_Change_Form(design, source_code)
67         IF fix_code IS NULL;
68     More_Code_Change_Form(design, source_code, fix_code_feedback)
69         IF fix_code = TRUE;
     ...
70 END_TASK;

71 BASE_TASK Modify_Test_Plans
     ...
72 END_TASK;

73 BASE_TASK Modify_Unit_Test_Package
     ...
74 END_TASK;

75 BASE_TASK Test_Unit
     ...
76   SCHEDULE
77     FOR AT(97/7/1,9:00:00) LASTING INTERVAL(0/0/7);
78 END_TASK;

79 END_MODEL
```

Figure 8. Base Tasks Modelling the ISPW-6 Software Process Example

The control part, line 6-8, specifies the top level activity abstraction of the workflow application. The workflow is supposed to start from the "Schedule_Task" (line 6) and finish with the "Perform_Change" task (line 8). The transition from one task to another is captured using the ENABLE statement (line 7 shows its simplest form). Conditional transitions can be specified by providing a condition as in line 22.

Three exception handlers are defined in line 10-18. "Test_Unit_Delayed" informs the project leader if "Test_Unit" is not started after the scheduled time. AT is a time operator which compares the given time with the current system time while STARTED is a pre-defined predicate indicating if a task has begun. "Test_Unit_Overdue" does the same when "Test_Unit" is not completed by its scheduled time. ELAPSED returns the time interval between the start of a task and the current system time. "Review_Design_Completed" simply informs the project leader of the completion of "Review_Design" together with information about the defects. COMPLETED checks if the given task is finished.

"Perform_Change" is further refined between line 19-33 and is specified to have two starting tasks (line 20). Line 21-27 concern the modification of design and code while line 28-31 deal with the testing of the modified code. A conditional transition is used in line 22 to allow "Modify Design" to be performed repeatedly until the design is considered acceptable by "Review_Design". "Modify_Code" (line 26) can be initiated in two ways: (1) after "Modify_Design" is started (line 24), (2) after "Test_Unit" is finished and suggests more code change (line 25). Line 27 asserts that "Modify_Code" cannot terminate before "Modify_Design" has done so. Task names that appear on the left of an ENABLE statement are assumed to signify the completion of the tasks. While those on the right are assumed to signify the beginning of the tasks. When a list of tasks appears on the left hand side of ENABLES without qualification (line 31), they all must hold before the task on the right hand side can start. STARTED (line 24) and COMPLETED (line 27) allow delicate control to be specified. In this case, they are used to specify that "Modify_Design" and "Modify_Code" are not ordered sequentially but instead can be overlapped.

In the case of a base task, instead of control statements, computation and message display are specified. Forms are displayed only by base tasks which have the duty of performing computation that may require inputs from the user. The forms to be displayed is specified in the VIEW clause and their contents are filled using local variables (e.g. line 36). The mechanism is similar to pass by reference in procedure calls in programming languages.

Every activity can be given a pre-condition and a post-condition that have to be satisfied for it to begin and end successfully. The pre-condition of "Schedule_Task" given in line 37-39 specifies that the two fields in the form must be blank. The post-condition given in line 43 specifies that all the fields must be filled. The keyword EVERY is a shorthand for all the fields in a form. The role constraint given in line 44-45 states that the task must be carried out by the project leader. It is also possible to use more than one form in a base task. Two forms are specified in "Modify_Design" (line 48-52). One form (line 49-50) is used when it is initiated for the first time and the other form (line 51-52) is used when further change to the design is required. These forms are used only if their associated conditions hold (line 50, 52). A base task can be assigned to multiple agents and this is demonstrated in line 60-62. Scheduling constraints can be given like in line 77 which states that "Test_Unit" must start by 9 am on July 1 and should be completed in 7 days.

5 System Architecture

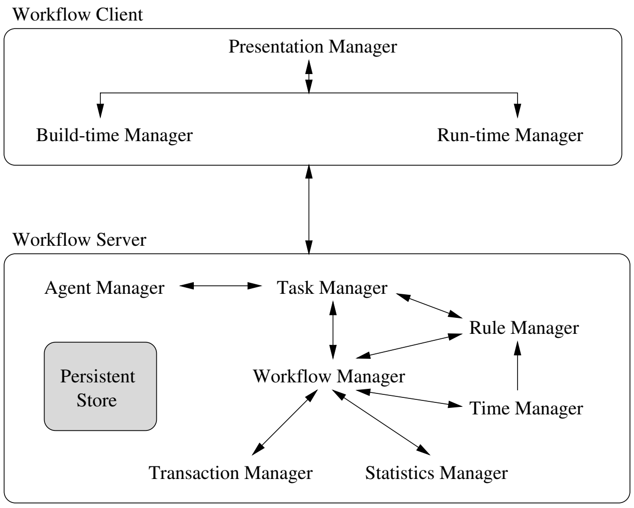

Figure 9. System Architecture

The architecture that supports *Valmont* is sketched in Figure 9. The system is implemented using OPJ (Orthogonal Persistent Java) [2] which provides orthogonal persistence [3], distribution support [20] as well as transaction support [9]. With OPJ, data of any type including procedures can be made persistent by attaching to a persistent root. There is no need to explicitly translate run-time data structures to persistent data structures such as to a database. An extended form of the Uniform Resource Locator (URL) called Persistent Object Locator (POL) is used to globally name any object in any persistent store. Conditions can be associated to objects to control their access and usage from remote locations. Transaction primitives are provided as OPJ object classes which can be used by expert users to build application specific transaction models. Such introduction of transactional behaviour can even be done dynamically without requiring re-building of the system or its applications. Less experienced users can define transactional applications by simply using the already defined transaction classes.

The system supports a client-server architecture. A client process can be either a build-time process or a run-time process. The former is controlled by the *build-time manager* which deals with the definition of various models and their consistency checks. The latter is controlled by the *run-time manager* which initiates the run-time support for the execution and monitors its progress. The two managers interact with the user via a *presentation manager*.

The *workflow manager* governs the workflow system on the server side and is primarily responsible for the scheduling of tasks. A scheduled task is then managed by the *task manager* which ensures its pre-conditions are satisfied before executing the task and its post-conditions are satisfied before termination. Task assignment is carried out by the *agent manager* on its behalf. The latter takes into account all assignment constraints to ensure optimal assignment. Exception handlers are dealt with by the *rule manager* with support from the *time manager* for temporal behaviour. The *statistics manager* is responsible for the gathering of management information. All the managers make use of the persistent store but this fact is not depicted in Figure 9.

6 Conclusions

This paper presented a user-oriented workflow language called *Valmont*, which differs from existing workflow languages and proposals in that all fundamental elements of the workflow paradigm are captured and elegantly integrated. Distinctive features include use of possibly partially parameterised domains, combination of specifications, incorporation of presentation details in form specifications, reusability over form specifications, declarative activity assignment, declarative exception handling, and workflow transactions. The expressive power of *Valmont* was demonstrated by providing a solution of the ISPW-6 Software Process Example which is a well known test-case for software process modelling. *Valmont* is being developed using OPJ as the implementation language and using web browsers such as Netscape as its front-end for user interaction.

References

[1] G. Alonso, D. Agrawal, A. El Abbadi, M. Kamath, R. Günthör, and C. Mohan. Advanced Transaction Models in Workflow Contexts. In *Proceedings of the International Conference on Data Engineering*, pages 574–581. IEEE Computer Society Press, 1996.

[2] M. Atkinson, L. Daynès, M. Jordan, T. Printezis, and S. Spence. An Orthogonally Persistent Java. *ACM SIGMOD Record*, 25(4):68–75, 1996.

[3] M. Atkinson and M. R. Orthogonal Persistent Object Systems. *VLDB Journal*, 4(3):319–401, 1995.

[4] E. Best and C. Fernandez. *Nonsequential Processes - A Petri Net View*, volume 13 of *EATCA Monographs on Theoretical Computer Science*. Springer-Verlag, 1988.

[5] C. Bussler. Policy Resolution in Workflow Management Systems. *Digital Technical Journal*, 6(4):26–49, 1994.

[6] F. Casati, P. Grefen, B. Pernici, G. Pozzi, and G. Sánchez. WIDE Workflow Model and Architecture. Technical Report 96-19, Centre for Telematics and Information Technology (CTIT), University of Twente, Netherlands, 1996.

[7] D. Chan and K. Leung. A Workflow Vista of the Software Process. In *Proceedings of the International Workshop on Database and Expert Systems Applications*, pages 62–67. IEEE Computer Society Press, 1997.

[8] D. Chan, J. Vonk, G. Sánchez, P. Grefen, and P. Apers. A Conceptual Workflow Specification Language. Technical Report 96-48, Centre for Telematics and Information Technology (CTIT), University of Twente, Netherlands, 1996.

[9] L. Daynès. Extensible Transaction Management in PJava. In *1st International Workshop on Persistence and Java*, 1996.

[10] J. Eder and W. Liebhart. The Workflow Activity Model WAMO. In *Proceedings of the 3rd International Conference on Cooperative Information Systems*, pages 87–98. US West Advanced Technologies, 1995.

[11] C. Ellis and G. Nutt. Modelling and Enactment of Workflow Systems. In *Application and Theory of Petri Nets*, volume 691 of *Lecture Notes in Computer Science*, pages 1–16. Springer-Verlag, 1993.

[12] A. Forst, E. Kühn, and O. Bukhres. General Purpose Workflow Languages. *Distributed and Parallel Databases*, 3(2):187–218, 1995.

[13] D. Harel. Statecharts: A Visual Formalism For Complex Systems. *Science of Computer Programming*, 8(3):231–274, 1987.

[14] IBM Corporation, U.S.A. *Modelling Workflow*, 1996. SH19-8241-01.

[15] K. Jensen and G. Rozenberg, editors. *High-Level Petri Nets: Theory and Applications*. Springer-Verlag, 1991.

[16] M. Kellner, P. Feiler, A. Finkelstein, T. Katayama, L. Osterweil, M. Penedo, and H. Rombach. ISPW-6 Software Process Example. In *Proceedings of the International Conference on the Software Process*, pages 176–186. IEEE Press, 1991.

[17] N. Krishnakumar and A. Sheth. Managing Heterogeneous Multi-System Tasks to Support Enterprise-Wide Operations. *Distributed and Parallel Databases*, 3(2):155–186, 1995.

[18] R. Medina-Mora, T. Winograd, R. Flores, and F. Flores. The Action Workflow Approach to Workflow Management Technology. In *Proceedings of the ACM Conference on Computer Supported Cooperative Work*, pages 281–288. ACM Press, 1992.

[19] T. Schäl. *Workflow Management Systems for Process Organisations*, volume 1096 of *Lecture Notes in Computer Science*. Springer-Verlag, 1996.

[20] S. Spence and M. Atkinson. A Scalable Model of Distribution Promoting Autonomy of and Cooperation between PJava Object Stores. In *30th Hawaii International Conference on System Sciences*, pages 7–10, 1997.

[21] R. Wagner, editor. *Proceedings of the International Workshop on Database and Expert Systems Applications (Workflow Management in Scientific and Engineering Applications)*. IEEE Computer Society Press, 1997.

[22] Interface 1: Process Definition Interchange. Technical Report TC-1016, Workflow Management Coalition, May 1996.

[23] Terminology & Glossary. Technical Report TC-1011, Workflow Management Coalition, June 1996.

[24] T. White and L. Fischer. *New Tools for New Times: the Workflow Paradigm*. Future Strategies Inc., 1995.