Liaison: a Workflow Model for Novel Applications

Daniel K.C. Chan◇ * Karl R.P.H. Leung□§ Ching Ying Yan□ Keith C.C. Chan□§

◇Project VERSO, INRIA - Rocquencourt, Domaine de Voluceau,
B.P. 105, 78153 Le Chesnay Cedex, France
Email: Daniel.Chan@inria.fr, Tel: +33 1 39 63 52 66, Fax: +33 1 39 63 56 74

□Department of Computing, Hong Kong Polytechnic University, Hong Kong
Email: {cskleung,cscyyan,cskcchan}@comp.polyu.edu.hk
Tel: +33 1 39635674, +852 2766 7267 Fax: +33 1 39635266, +852 2774 0842

Abstract

The workflow paradigm is advancing the scope of process modelling by supporting rich process structures as well as resource allocation through the use of an organisation model. Its initial success has attracted a lot of interests in applying it to applications that are more complex than what can be handled by the current generation of workflow management systems. This paper introduces a new workflow model called Liaison that is designed to address the needs of these novel applications. More specifically, it provides a rich organisation model, sophisticated activity assignment constraints, an information model that also records presentation details, and a dynamic process model. An implemented prototype is described together with an examination of desirable improvements.

1 Introduction

The workflow paradigm is advancing the scope of process modelling by supporting rich process structures as well as resource allocation through the use of an organisation model. It has enjoyed some success in business process re-engineering [20] and consequently is attracting a lot of attention from other application domains such as scientific and engineering applications [19] as well as the software process [5]. In many of these domains an application is such a complex activity that automated tools for the management and the execution of the process are highly desirable. It is precisely because of the fundamental ability of workflow management systems to model, co-ordinate, and monitor complex activities that attempts are being made to extend the paradigm with new features to support more advanced applications.

This paper presents a workflow model called *Liaison* that is designed to address the needs of novel applications. Unlike many other workflow models, *Liaison* provides a rich organisation model that allows incomplete information to be captured and analysed, an information model that also records presentation details, and a dynamic process model. Moreover, the assignment constraints between tasks in a process and agents in an organisation model are very sophisticated and more powerful than those provided by other workflow systems.

The rest of the paper is structured as follows. Section 2 presents a running example that is used throughout the paper. Section 3 gives an overview of *Liaison*. Section 4 focuses on the organisation modelling aspect of *Liaison*. Section 5 examines the modelling of information and their representation in *Liaison*. Section 6 investigates the essence of process modelling. Section 7 sketches the implementation of a prototype of *Liaison* and discusses the possibilities of improvement. Section 8 concludes. Due to space restriction, the reader are referred to the discussion given in [6] for related work.

2 Running Example

The running example is actually part of a workflow system supporting the software development life cycle of a software house in Hong Kong. The implemented system supports several processes in the development cycle, namely system design, system testing, user acceptance test, change control, and project management. Figure 1 presents

*Supported by a Marie Curie Fellowship from the European Commission (ERBFMBICT960653). §Supported by research grants from the Hong Kong Polytechnic University (GV-639 and TCS 745).

only the system design process.

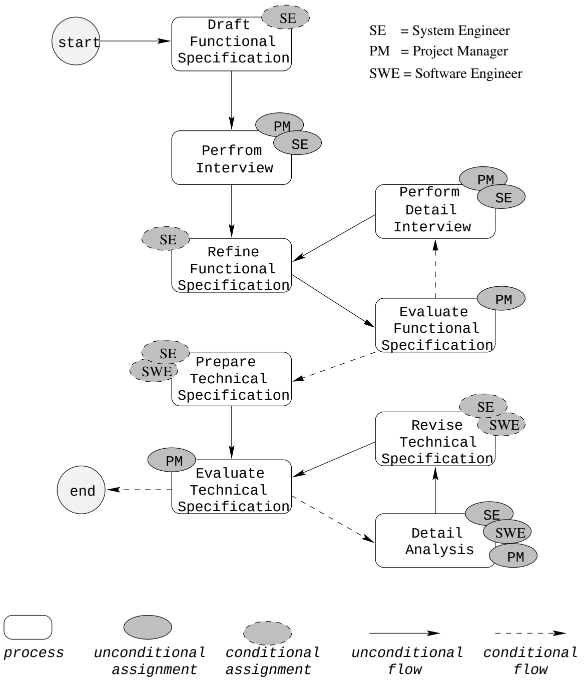

Figure 1. System Design Process

The design process starts with the drafting of the functional specification (FS) whose "correctness" is checked through interviews with the user, for instance a Unit Manager responsible for the application that is being automated. Feedback from the interviews leads to refinement of the functional specification. It is then evaluated to decide if more interviews need to be performed and changes to be included. When a satisfactory functional specification is obtained the technical specification (TS) is prepared. Similarly, the technical specification is subject to a evaluation process. An unfavourable conclusion may result in a detail analysis being performed followed by a revision of the specification. The project team consists of staff members with the capacities of Project Manager (PM), System Engineer (SE), and Software Engineer (SWE). Besides the two specifications, other documents are also used in the process and they are referred to as discussion records and working papers.

3 An Overview of Liaison

The *Liaison* workflow model is an extension of the reference workflow model proposed by the Workflow Management Coalition [16]. 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, and a *form model* concerning the presentation and access control of data. The examples used in this paper are expressed using *Valmont* [6], which is a textual representation language of *Liaison* and a descendent of a language developed earlier by the first author [7]. A summary of the "sub-models" of *Liaison* is given next and more details are revealed in the coming sections.

3.1 Organisation Modelling

The very first stage of every application development project would be to approach the personnel related to and affected by the project so as to understand the general objectives, requirements, and operation environment. This often translates to figuring out the organisation hierarchy and interviewing senior personnel before interviewing less senior personnel. The organisation hierarchy can be captured as a position hierarchy.

It should be noted that such a hierarchy captures the delegation and accountability relationships but does not usually reflect the functional and operational structures of an enterprise. However, as far as workflow application design is concerned, in many cases, the operational structure of an activity can be constructed out of the position hierarchy. In order to maximise the reusability of the position hierarchy and to minimise the effort required to model the operational structures, an elaboration of the lower levels of the position hierarchy is encouraged. Relatively speaking, a less refined account of the upper levels would suffice.

The functional structures of an enterprise can be captured using teams and the team hierarchy. The grouping of individuals into teams should also take activity assignment into consideration. For example, a large team would be preferred for high volume or repetitive activities. It should be noted that it is possible and likely that the team hierarchy contains free standing teams that are not be related to any other teams (see the team hierarchy on the right of the abstract organisation structure shown in Figure 2). In other words, the team hierarchy is more likely to be a "forest" rather than just a "tree".

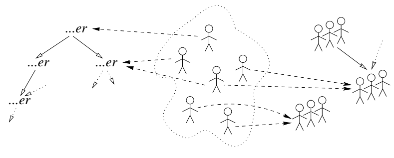

Figure 2. Positions, Staff, and Teams

The relationships between positions, individuals, and teams are shown in Figure 2. The position hierarchy is shown on the left where a *...er* stands for a position such as Information Technology Manag*er* or Programm*er*. The assignment of individuals to positions and to teams is shown by the dotted line arrows.

3.2 Information Modelling

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 presentation of data. In other words, the model defines data that are passed between activities within a workflow process as well as data that go across the boundary of the workflow process. The first kind of data is referred to as *local* data which can be application data or control data. They are managed by the workflow management system. The second kind of data is referred to as *external* data which serve as connections between the workflow process and other external systems that may be heterogeneous and distributed in different geographical locations. For instance, the system design process described before may need to access a database containing templates for system testing, it may also invoke another application for the purpose of processing the functional specification, or it may need to communicate with an existing workflow process dealing with the interview procedure as shown in Figure 3. A piece of such external data is represented in *Valmont* as an *entity* whose state may not be known to the workflow systems but usually operations are defined to manipulate the data. Both local and external data are defined in the *data model* which is a sub-model of the information model.

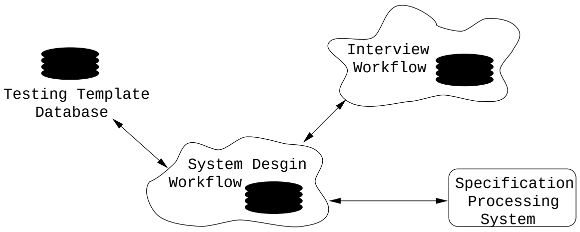

Figure 3. Data Access of a Workflow Process

Presentation of data is defined independently in another sub-model called the *form model* [4]. A form is a unit of presentation that is used whenever interaction with the user is required. It is possible that several forms can be displayed for a single task. Typically a form has a number of fields in which data can be displayed and captured. To improve quality of the display, formatting information can be provided. One form can be derived from another form possibly adding new fields and restricting operations that can be applied on some fields. This mechanism encourages re-usability which, in turn, reduces the maintenance effort.

3.3 Process Modelling

The *process model* plays a central role among the sub-models of *Liaison*. It ties the organisation, data, and form models to activities representing the steps of the application being modelled. To facilitate the modelling of complex systems as well as making the resultant design more comprehensible, the process model supports incremental modelling allowing an activity to be refined into more detailed activities (see Figure 4). In other words a typical design would end up with several layers of activities with more abstract activities called *tasks* at the upper layers and more concrete activities called *base tasks* appearing in larger number at the very bottom layer.

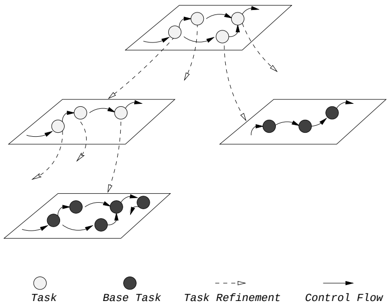

Figure 4. Task Refinement

The concern of the first stage of workflow process modelling is to identify the concerned agents and figure out the activities they have to carry out to fulfill the application. The identified agents should have been captured in the organisation model. Activities and their order of execution (i.e. *control flows*) together with the responsible agents (i.e. *activity assignment*) are then captured in the process model. Since both control flows and activity assignment can have conditions associated with them, different notations can be used to reflect the situation (like those shown in Figure 1). Abnormal situations in the process can then be identified as *exceptions* which may be triggered by activity events that happen in the process as well as temporal events that are defined in the process. For instance, it is possible to raise an exception when no agent can be found to carry out an activity or to send a reminder to the Project Manager when the review of the functional specification is due, say in three days.

So far what has been discussed can be applied to both tasks and base tasks; however, base tasks can perform other actions that are not found in tasks. The primary difference is the possibility of displaying forms (described in the form model) to initiate some interaction with the user. It is not unusual that the data and form models need to be revised during process modelling.

4 Organisation Model

The organisation model registers the structure and resources of an organisation. It records information about individual employees (*staff*), functional positions held by staff (*position*), and groups of staff (*team*) that are put together to serve some transaction. 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, and teams are collectively referred to as *agents*.

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. 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 *parent* team which includes the original team can be brought in.

Liaison has a set of pre-defined attributes for staff members; however, user-defined attributes can be added to address the specific needs of enterprises and applications. The example given in Figure 5 shows how two attributes capturing the programming language and system skills of staff members can be introduced.

```
STAFF HAS
  STRING[] language, system;
END_STAFF
```

Figure 5. User-defined Staff Attributes

As a consequence of the definition given in Figure 5 the pre-defined attributes of staff members are augmented resulting in a "full" definition like that shown in Figure 6. The "AVAILABLE" attribute records if a staff member is available while the "DEPUTY" attribute specifies a list of staff members who can stand in for the staff member.

```
STAFF HAS
  STRING NAME, LOGIN, SERVER;
  INT PHONE;
  BOOLEAN AVAILABLE;
  STAFF[] DEPUTY;
  ...
  STRING[] language, system;
END_STAFF
```

Figure 6. Staff Attributes

To record a staff member identified as "Andy" with the attributes given in Figure 6, the REGISTER statement as shown in Figure 7 can be used (a pair of square brackets indicates a list value). Note that certain attributes (e.g. "PHONE") may be left un-defined as a result of not being given a value in the REGISTER statement. However, some attributes such as "AVAILABLE" are initialised to their default values when none is given in the registration.

```
REGISTER STAFF CALLED Andy HAVING
  (NAME="Andy", LOGIN="aLee", SERVER="csun50.comp.polyu.edu.hk",
   DEPUTY=[Billy], language=["Java", "C++"], system=["UNIX", "DOS"]);
```

Figure 7. Staff Registration

The way positions are defined is very similar to that for staff members. Figure 8 contains two position definitions: (1) "Software_Engineer" with attribute "authorisation", "SUPERIOR" which is a pre-defined attribute, and "system" which captures the system skills of a software engineer, (2) "Project_Manager" with no attribute. The attribute "SUPERIOR" given in line 3 establishes the fact that a "Software_Engineer" is accountable to a "Project_Manager" who is however accountable to nobody as no "SUPERIOR" attribute is given in line 7. In other words, if no "Software_Engineer" is available when one is required a "Project_Manager" will be called in. The attribute "system" has an intensional constraint asserting that it *should* contain between 3 and 5 elements. Note that this intensional constraint serves to capture *incomplete* information. In other words, one can register a staff as a software engineer without giving any of his or her system skills (see line 8-11). The registered software engineer is therefore known to the organisation model as having certain number of system skills but the exact information of these skills is not available at the time of registration. This is a very practical concept since a large number of positions can be registered without precise information.

```
1  POSITION Software_Engineer HAS
2    INT authorisation;
3    STAFF SUPERIOR BEING Project_Manager;
4    STRING[] system WITH AT_LEAST 3 ELEMENTS
5                    WITH AT_MOST 5 ELEMENTS;
6  END_POSITION

7  POSITION Project_Manager END_POSITION
   ...
8  REGISTER POSITION Software_Engineer HAVING
9    (HOLDER=Andy, SUPERIOR=Chris, authorisation=3);

10 REGISTER POSITION Project_Manager HAVING
11   (HOLDER=Chris);
```

Figure 8. Position Definition & Registration

Organisations are becoming more team-oriented rather than function-oriented as the former helps eliminate unnecessary control resulting in better responsiveness and flexibility. Teams can be defined using the TEAM definition as shown in Figure 9. Teams can form a team hierarchy using the inclusion relationship. In this case, line 1 states that "Testing_Team" is included in "Project_Team". It is also possible to specify the capacities of members in a team. For example, line 6 and 7 state that "Project_Team" has to have at least two software engineers and at least two system engineers.

```
1   TEAM Testing_Team WITHIN Project_Team HAS
2     STRING[] machine;
3   END_TEAM

4   TEAM Project_Team HAS
5     STAFF[] members WITH AT_LEAST 2 ELEMENTS BEING System_Engineer
6                     WITH AT_LEAST 2 ELEMENTS BEING Software_Engineer;
7   END_TEAM
    ...
8   REGISTER TEAM Testing_Team HAVING
9     (LEADER=Andy, machine=["PC", "PC", "SPARC"]);
    ...
10  REGISTER TEAM Project_Team HAVING
11    (MEMBER=[Andy, Billy BEING Software_Engineer, Chris]);
```

Figure 9. Team Definition & Registration

Every team has a number of pre-defined attributes including "LEADER" and "MEMBER". Line 9 of Figure 9 names "Andy" as the leader of the "Testing_Team". Since a team member can hold more than one position, it is sometime necessary to specify the capacity of a team member. In line 11, "Billy" serves in the "Project_Team" as a "Software_Engineer" rather than the other capacities (i.e. "System_Engineer") that he may hold.

5 Information Model

5.1 Data 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. Currently only simple data types and bounded list type are supported together with the *entity* type which is introduced to capture external data and their associated operations. Every entity is identified by a unique identifier in the global context and may have a set of associated operations. The data model shown in Figure 10 contains two categories of data. Line 2-5 capture an external document as an entity and defines three associated operations: "read", "edit", and "print".

5.2 Form Model

A form resembles a record except that it also contains presentation information which is important due to the interactive nature of workflow systems - a lot of information

```
1  DATA_MODEL System_Design_Process
2    ENTITY functional_spec
3    IDENTIFIED BY "/stfc/project915/design.txt";
4      HAS read, edit, print;
5    END_ENTITY;
     ...
6    VARIABLE
7      STRING fs_status, ts_status;
8  END_MODEL
```

Figure 10. The Data Model (Incomplete)

is presented and captured using forms displayed on user terminals. Presentation information include headings, field labels, and formatting information. Each field can be used to display or update data variables defined in the data model. Default field values can also be specified. A form can be derived from another form possibly by adding some fields. Moreover a derived form can restrict operations that can be performed on a field.

A number of forms can be defined in a form model. For instance, five forms are required for the running example but only two are shown in Figure 11. The "FS_Form" defined in line 2-6 is to allow access to the functional specification which is assumed to be a string of text. The "HEADING" clause (line 3) prints a string on the screen while SPACE reserves some vertical spacing on the screen. Each LABEL clause can generate a box for the display or capture of the corresponding data. For example, line 5 generates a box next to the label "Functional Specification". It should be noted that fields in a form are references to data defined in the data model and they are not value holders themselves. Type information of the fields are included to facilitate type checking and efficient formatting.

The "TS_Evaluation_Form" is derived from the "FS_Form" and contains the additional fields: "ts" and "status". An initial value is given to "status" in line 11. Access constraints over fields in the form are given in line 12 stating that "fs" and "ts" can only be read but not updated.

```
1   FORM_MODEL System_Design_Process
2     FORM FS_Form
3       HEADING "Functional Specification";
4       SPACE;
5       LABEL "Functional Specification" LINKS STRING fs;
6     END_FORM
      ...
7     FORM TS_Evaluation_Form DERIVED_FROM FS_Form
8       LABEL "Technical Specification" LINKS STRING ts;
9       SPACE;
10      LABEL "Recommendation" LINKS STRING status;
11      INITIALLY status="";
12      THROUGHOUT fs READ ONLY, ts READ ONLY;
13    END_FORM
14  END_MODEL
```

Figure 11. The Form Model (Incomplete)

6 Process Model

The process model not only specifies the activities of a workflow process but also integrates the other sub-models defined for the process. The workflow process definition given in Figure 12 begins with importing the organisation model (line 2), data model (line 3), and form model (line 4).

The rest of the process specification include the control flow specification (CONTROL section), activity assignment constraint specification (ROLE section), exception handling specification (EXCEPTION section), and the activity refinement specification which, in turn, includes refinements for *tasks* (TASK sections) as well as definitions for *base tasks* (BASE_TASK sections). The control section, line 5-12, specifies the control flows between activities in the workflow process. It begins with "Draft_FS" (line 6) and finish with "Evaluate_TS" when the data "ts_status" is set to "completed". The control flow from one activity to another is captured using the ENABLE clause. Line 7 shows its simplest form in which one activity enables another activity. More complex control flows, such as various forms of *fork* and *join*, can be specified using qualified ENABLE clauses (i.e. "TOGETHER ENABLE", "ENABLES ANY integer", "AND", and "OR"). Conditional control flows can be specified by providing a condition in a "IF" clause as demonstrated in line 10-11.

```
1  WORKFLOW_MODEL System_Design_Process

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

5  CONTROL
6    START Draft_FS;
7    Draft_FS ENABLES Perform_Interview;
8    Perform_Interview ENABLES Refine_FS;
9    Refine_FS ENABLES Evaluate_FSw;
10   Evaluate_FS IF fs_status = "to be revised" ENABLES Perform_Detail_Interview;
11   Evaluate_FS IF fs_status = "completed" ENABLES Prepare_TS;
     ...
12   END Evaluate_TS IF ts_status = "completed";

13 ROLE
14   Draft_FS, Refine_FS BY SAME STAFF;

15 EXCEPTION
16   WITHIN 000300 BEFORE Evaluate_FS.due_date
17     DOES MESSAGE "Review will be due in three days.";

18 BASE_TASK Evaluate_FS
19   VIEW FS_Evaluation_Form(fs = functional_spec, dr = discussion_records,
            wp = working_papers, status = fs_status);
20   MESSAGE "Please recommend if the functional specification is acceptable.";
21   POST_CONDITION fs_status = "to be revised" OR fs_status = "completed";
22   BY STAFF BEING Project_Manager IN Project_Team;
23 END_TASK;
   ...
24 END_MODEL
```

Figure 12. The Process Model (Incomplete)

It should be noted that control flows are defined using activity *states*. The possible states and the *events* causing transitions between states are shown in Figure 13. An activity is in the *created* state when it is created together with all its statistics and resources. It enters the *started* state when the pre_condition is met. It then assumes the *running* state when an agent is assigned to it. An activity can also be suspended or rejected for various reasons. It can also be aborted at any time. A successful execution enters first the *satisfied* state when the post_condition is satisfied then the *ended* state when all housekeeping is done. Activity names that appear on the left of an ENABLE clause are assumed to signify the completion of the activities (i.e. of *ended* state). While those on the right are assumed to signify the beginning of the activities (i.e. of *started* state). When a list of tasks appears on the left hand side of ENABLES without qualification, they all must be completed before the tasks on the right hand side can start.

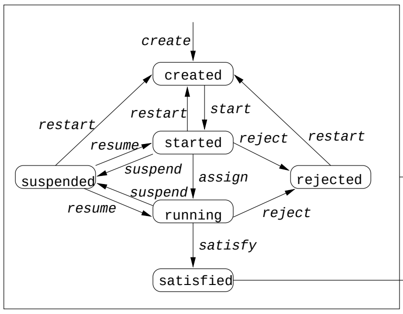

Figure 13. Process States & Transitions

All activities must have one and only one refinement or definition section. A task refinement section is very similar to a workflow definition. It consists of a control flow section, an activity assignment constraint section, and an exception handling section. While a base task definition is all together different. It consists of a form display section, a pre-condition section, a message display section, a computation section, a post-condition section, and an activity assignment constraint section. The definition of "Evaluate_FS" is given between line 18-23. It displays the form "FS_Evaluation_Form" using "functional_spec", "discussion_records", "working_papers", and "fs_status" to fill in the fields in the form (line 19). It also displays a message to the user stating the purpose of the activity (line 20). A post_condition requiring the "fs_status" variable to be set to either "to be revised" or "completed" is imposed (line 21). In effect, the user must enter either one of the two options in "FS_Evaluation_Form" before the activity can be terminated successfully. Actually a pre_condition can also be specified before the "VIEW" clause and it must be satisfied before the activity can execute. The keyword "EVERY" is a shorthand for all the fields in a form. So if all the fields must be filled when the activity ends a post_condition can be specified as "EVERY FS_Evaluation_Form IS NOT NULL".

Activity assignment constraints can be specified for both task and base tasks. The constraint given in line 13-14 states that "Draft_FS" and "Refine_FS" must be performed by the same staff meaning effectively the same "System_Engineer" must be responsible for the revision of his or her own draft. The constraint given in line 22 states that the activity is to be performed by a "Project Manager" who works in the "Project_Team". More complex assignment constraints involving attribute values and relationships between STAFF, POSITION, and TEAM can be specified together with the use of intensional constraints discussed earlier. Note also that an activity assignment constraint in a task definition specifies dependency between activity assignments of lower-level activities. While that of a base task is always about the base task alone.

The control flow section of an application models the flows that normally happen between activities. Situations that are considered "abnormal" are captured as exceptions. Such exceptions can be raised as the result of a number of possible events. Some events are temporal in nature such as the scheduled deadline of an activity. Others are events concerning the behaviour of the workflow system such as the cancellation of a task or when no agent can be found for a task. Indeed, exceptions can be found in all sub-models of *Liaison* as well as the relationships between the sub-models. Exceptions can be handled by defining exception handlers in the process model. An example is given in line 15-17 and it displays a message when "Evaluate_FS" is due in three days.

7 Prototype

7.1 Operation of the Prototype

A prototype of *Liaison* has been implemented to support a subset of the features described in the previous sections. It provides a graphical editor that can be used to define or modify the sub-models of an application. For instance, the process model of the running example can be defined using the graphical editor with the result shown in Figure 14.

When the application is being defined, dialog boxes are displayed from time to time to capture various details that cannot be captured using the drawing tools. One such detail is the activity assignment constraint of a task. The view shown in Figure 15 displays the agents involved in an application and a line is drawn between two agents if they are co-responsible for the execution of some task.

Different views are provided for the completed specification. Details of the application and each activity can also be displayed (e.g. Figure 16).

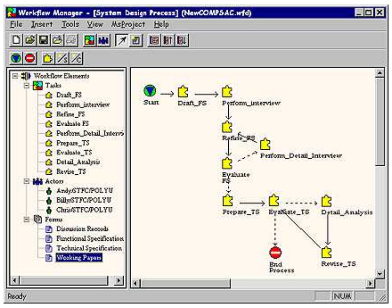

Figure 14. Process Model

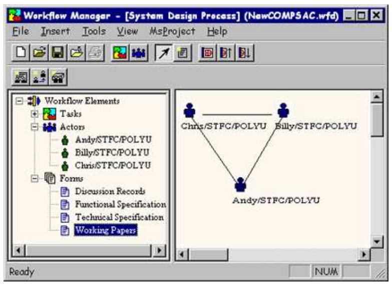

Figure 15. Organisation Model

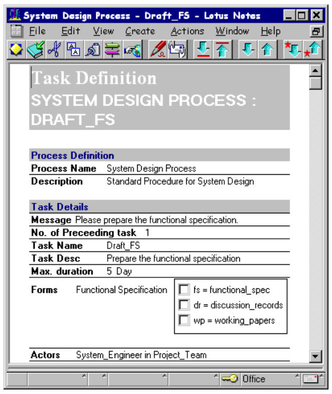

Figure 16. Task Definition Display

7.2 Limitations

The prototype is implemented using Visual C++ on top of Lotus Notes. A number of the limitations of the prototype are the direct results of using Lotus Notes as the underlying engine. At the moment, the implemented organisation model supports only the organisation hierarchy but not teams. Further development is undertaking to improve the model to support teams. The implemented form model is based on *forms* supported by Notes. It therefore supports incorporation of multimedia objects but in general external data access will require additional treatment. The prototype does not support task abstraction resulting in a process model having only a single layer of activities. Extensions along this line may be problematic and possible solutions are still being investigated. Exceptions are user-driven rather than system-driven and we are not very optimistic in extending the prototype to facilitate system-driven exceptions. Nevertheless, the prototype can still support a reasonable number of applications picked from the software house previously mentioned even without the non-implemented features.

8 Conclusions

This paper presented a new workflow model called *Liaison* which distinguishes itself from other workflow systems in that it supports a complex organisation model and captures presentation information in the form model. These extensions are important for scheduling of tasks and interacting with the user. These are parts of our continuing effort to develop a workflow system that meets the need of novel applications.

Application modelling using *Liaison* is briefly discussed and is followed by a comprehensive examination of its features. The organisation model supports the notions of positions and teams that are very useful in capturing the operational structure of an organisation. The form model provides a bridge between the user and the application. Through forms, the user can update data defined in the data model and consequently affect the behaviour of the system. The process supports complex control flows, exceptions, and sophisticated activity assignment constraints.

A prototype implemented on top of Lotus Notes has proved to be rather limited and difficult to extend to incorporate all the designed features. A new Java based implementation is being carried out to support all the features described in this paper. At the point of writing, three components have a new prototype running. The form model also has the capability of generating HTML pages automatically. The new system will meet the requirements of more applications than the current implementation even though the latter has shown to be quite sufficient for a number of applications.

References

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

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

[3] C. Bussler. Analysis of the Organisation Modelling Capability of Workflow Management Systems. In *Proceedings of the PRIISM '96 Conference*, pages 438–455. Springer-Verlag, 1996.

[4] D. Chan. Form Management in the Vicomte Workflow System. In *Proceedings of the 32nd Hawaii International Conference on System Sciences*, 1999.

[5] D. Chan and K. Leung. A Workflow Vista of the Software Process. In *Proceedings of the 8th International Workshop on Database and Expert Systems Applications*, pages 62–67, France, Sept. 1997. IEEE Computer Society Press.

[6] D. Chan and K. Leung. Valmont: a Language for Workflow Programming. In *Proceedings of the 31st Hawaii International Conference on System Sciences*, volume VII, pages 744–753, USA, Jan. 1998. IEEE Computer Society Press.

[7] D. Chan, J. Vonk, P. Grefen, and P. Apers. A Specification Language for the WIDE Workflow Model. In *Proceedings of the ACM Symposium of Applied Computing*, pages 197–199, USA, Feb. 1998. ACM Press.

[8] 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.

[9] 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.

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

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

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

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

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

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

[16] P. Lawrence, editor. *Workflow Handbook 1997*. John Wiley & Sons, 1996.

[17] 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.

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

[19] 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.

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