



## COVER SHEET

---

**This is the author version of article published as:**

*Streit, Alexander T. and Pham, Binh L. and Brown, Ross A. (2005) Visualization Support for Managing Large Business Process Specifications. In van der Aalst, Wil M. P. and Benatallah, Boualem and Casati, Fabio and Curbera, Francisco, Eds. Proceedings 3rd International Conference, Business Process Management, BPM2005 3649, pages pp. 206-219, Nancy, France.*

**Copyright 2005 Springer**

**Accessed from <http://eprints.qut.edu.au>**

# Visualization Support for Managing Large Business Process Specifications

Alexander Streit, Binh Pham, and Ross Brown

{a.streit,b.pham,r.brown}@qut.edu.au  
Faculty of Information Technology  
Queensland University of Technology  
2 George St, Brisbane Australia

**Abstract.** This paper proposes a visualization technique to support the modelling and management of large business process specifications. The technique uses a set of criteria to produce views of the specification that exclude less relevant features. The proposed approach consists of three steps: assessing the relevance of nodes, reducing the specification, and presenting the results. Algorithms and methods are presented for these steps along with examples.

## 1 Introduction

There are multiple graphical business process modelling techniques such as EPC (Event-driven Process Chain) and YAWL (Yet Another Workflow Language), for more see [1] pp.3. Graphical business process modelling languages are elegant solutions because the user can visually interpret the process. However, as the process grows in size the graph becomes difficult to deal with. This problem is well known to fields that use graphical languages [2]. While zooming initially solves the issue of gaining an overview perspective, there is a finite limit to the amount of zooming that can be performed. Screen real estate is limited and the specification given in Figure 7, for example, does not fit on a 1280x1024 display.

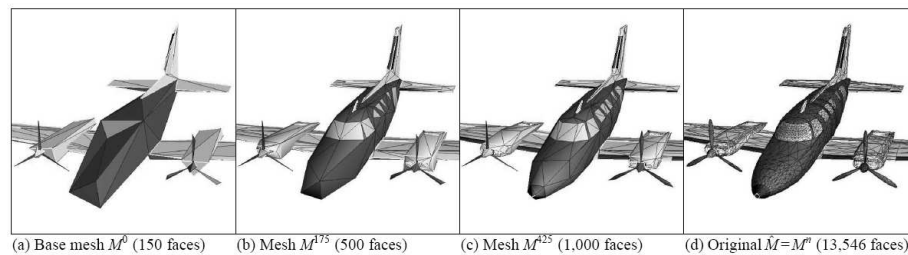
Features requiring controlled visual processing, such as interpretation of text, are dominant in business process modelling languages. The ability of controlled visual processing to be interpreted is particularly affected as more information is added. Automatic processing features, such as colour, find limited use in specification languages such as EPC and YAWL. In the case of EPC, where colour is used, colour does not contribute to the overall structural interpretation of the graph.

The traditional solution to this has been to allow decomposition of tasks to sub-specifications. This approach requires that the user construct a deliberate hierarchical structure to support what is in essence a multi-resolution model. Another approach is conversion of the information into another format, but this loses the benefits of user familiarity, requiring users to learn a new representation.

For large models to be understood it is necessary that the level of controlled processing required is reduced. The approach explored in this paper is to provide

views of the specification that exclude less relevant information. This filtering of information produces a model with lower complexity, but introduces a degree of uncertainty. This uncertainty reflects the lower resolution model's potential for representing variations of the original model. This use of uncertainty mimics human reasoning [3], where decisions are made on relevant information instead of relying upon a detailed and precise model.

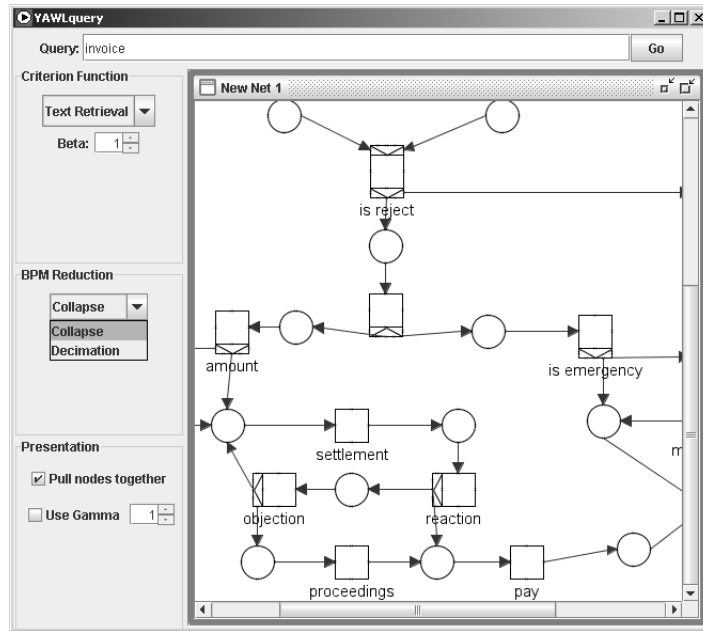
The discipline of 3D computer graphics has conducted extensive research into level of detail algorithms [4]. These algorithms construct simplified representations of a full scale model. The purpose of simplification is to maintain a representation of the model that is recognisable while reducing the processing and data requirements of the system (see Figure 1). Typically, lower level detail versions of a model are substituted for the object when it is further away from the observer, where the change is indiscernible.



**Fig. 1.** The structure of the 3D model of a plane is evident, even at four different levels of detail. (from [5])

The approach in this paper is motivated by the success of level of detail methods in the 3D graphics field. The proposal is a simplification approach for business process specifications by constructing a *reduced graph* that captures the most relevant information of the original graph. This technique avoids the intuitiveness issue mentioned previously by using the same graphical notation as the original graph. However, the reduced graph must also preserve the semantics of the original graph to avoid being misleading.

This reduction process presents an opportunity to not only preserve the overview of structure, but to actually provide different views of the same graph according to different interests of the user. Reduction should therefore be directed by criteria that represent the interest of the user, which is governed by the task of the user. For example, the user may wish to see only those processes that are involved in a possible dead-lock situation, or alternatively the user may wish to see nodes that are relevant to a text search term. A graphical search engine can be constructed by creating *reduced views* of business process models according to search terms. This effectively allows the user to browse the business process much like using a web search engine.



**Fig. 2.** Prototype for a system that allows users to query specifications in a similar manner to a web search engine.

To expand on the example of the search engine, consider the prototype<sup>1</sup> shown in Figure 2. The user is able to enter a search term that is used to direct the criterion function. The resulting display is a *reduced view* of the specification that includes only the most relevant nodes and their relationships to one another. Should the user enter a different term, the process is repeated, starting from the original specification every time. No changes are made to the original specification, instead a temporary reduced view of the specification is constructed for display to the user. Such a tool might be incorporated into the modelling package, to aid the user's understanding and construction of large or complicated specifications.

The mechanics of the reduction algorithm is based on first determining a relevancy factor for each node, followed by analysing the paths through the process model and removing the least relevant nodes. Once the graph has been reduced it must be prepared for display, which requires an aesthetically pleasing and intuitive layout for the graph.

Section 2 provides background material, section 3 details the techniques and approach, while section 4 provides a summary of the work and points to future work.

<sup>1</sup> More information will be made available through <http://www.bpmquery.com>

## 2 Background

### 2.1 Workflow specifications

Business process management (BPM) is about the management of business processes. BPM is receiving increased attention due to improvements in information systems [6]. Workflow management systems (WFMS) are computerised tools to support BPM and workflow specifications drive the WFMS.

Workflow specifications can be observed from different perspectives. The *control-flow perspective* describes the order of execution of tasks, that is it describes task dependency relationships. Tasks can either be atomic or decompose to sub-specifications, which creates a hierarchical view of the process. The *data perspective* deals with the flow of objects, such as documents, and can overlay the control flow perspective. The *resource perspective* links tasks to the resources required to perform them. The *operational perspective* details the practical execution of tasks, such as the underlying software services involved.

There are a number of workflow specification languages, both commercial and academic (see [1]). WF-nets were proposed [6] as a specification language based on Petri-nets. The advantage of using Petri-nets is that they provide a formal basis, which enforces precise definition. The disadvantage to this approach is that some patterns do not map well onto high-level Petri-nets [7]. YAWL [7] is a progression from WF-nets that overcomes these disadvantages by adding mechanisms to support the workflow patterns in [8]. The YAWL environment is freely available<sup>2</sup>.

The YAWL environment currently provides support for the control perspective, data perspective, and the operational perspective. The formal underpinnings and expressiveness of YAWL make it an ideal choice for visualization research. The former allows for a formal analysis of techniques, while the latter implies that successful development of techniques for YAWL will translate to other workflow specification languages.

The constructs of the YAWL language are given in Figure 3.

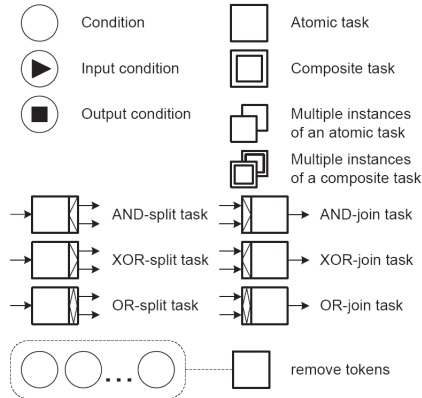
### 2.2 Visualization

A visualization program is analogous to a looking glass through which the user inspects an underlying system. In other words, it is the “bringing out of meaning in data” [9]. Examples of visualization techniques are given in [9]. Traditionally, visualization research has produced visualization techniques that were classified according to data type [10–13]. However, recent opinion has criticised this approach as producing “showy” images that are insufficiently useful to the user [14].

Suggestions for overcoming this include working more closely with the application domain [14] and creating task-oriented visualization systems [15]. Task-oriented visualizations are driven by the task of the user rather than the composition of the underlying data. The user-centric approach of task-oriented vi-

---

<sup>2</sup> The YAWL environment is available through <http://www.yawl-system.com>



**Fig. 3.** Constructs of the YAWL language [7]

sualization requires an understanding of the user’s requirements. This in turn requires closer cooperation with the application domain.

Visual elements can be classified into two categories [12]: automatic visual processing elements are easily interpreted and include colour, shape, and width, whereas controlled visual processing requires additional user interpretation and include features such as a text, icons, and arrows.

### 2.3 Mesh Simplification Algorithms

Real-time computer graphics applications use 3D mesh structures to model 3D objects. The mesh structure consists of a collection of convex surfaces defined by their vertices. The visible surfaces of the mesh are rasterized, to produced a raster image, which is subsequently shown to the user. To maintain interactive frame rates, this process must be performed for every visible 3D object, in under 83 milliseconds. The sheer mesh complexity required for acceptably accurate models creates processing challenges and has lead to the creation of novel techniques to reduce complexity.

Simplification algorithms reduce the mesh complexity while maintaining the important characteristics of the model. These techniques are used to reduce computation requirements for uses such as fluid flow simulation, shadow volume extrusion, and particularly preserving visual appearance. Several techniques exist (see [4]), which can be placed into two broad categories: *decimation* and *collapse*. Decimation techniques remove numerous elements and reconstruct surfaces over the holes this creates, whereas collapse methods incrementally reduce the mesh through atomic operations.

The progressive mesh [5] is a collapse technique designed for progressive transmission of mesh data. Partially received progressive meshes can be displayed to give the user a low resolution model and the model is refined as more data is received. The edge collapse technique used has an inverse operation, called a

vertex split. Given vertex split information, in the correct order, the mesh can be reconstructed to the desired level of detail.

All algorithms make use of an error metric to choose the appropriate reductions. The error metric varies depending upon the intended application of the simplified mesh. For example, the error metric used to generate the appearance preserving meshes given in Figure 1 uses an energy function that measures the squared distance of the proposed vertices to the original mesh, tempered by a spring function to distribute collapses across the mesh [5]. The interested reader is directed to [4] for detailed treatment of various error metrics.

## 2.4 Other Related Work

Researchers have previously identified comprehension issues with large conceptual schemas. Their solution builds abstractions for conceptual schemas through recursive derivation of simplified representations [16]. Each derived representation is termed an abstraction level. The abstraction mechanism introduces an importance rating for roles. Objects are weighted according to the sum of their anchored role weights. Object weights that exceed the current abstraction level threshold are identified as important and included in the abstraction level, whereas a series of production rules are used to remove the remaining objects and their associated roles.

## 3 Approach

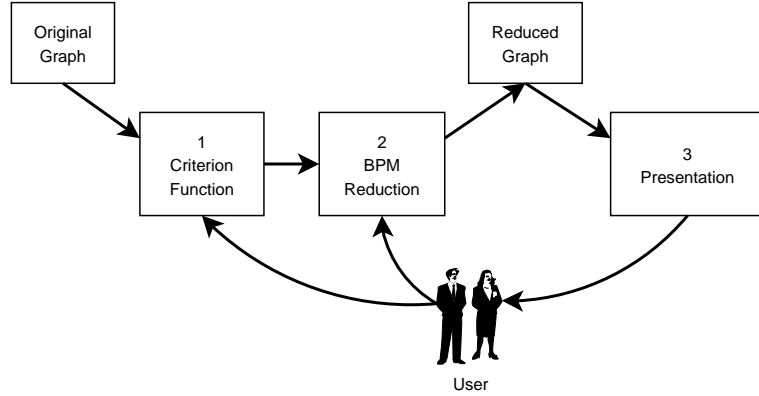
This section details the approach including the underlying algorithms. Section 3.1 describes methods for assessing the relevance of nodes, section 3.2 provides algorithms for reducing graphs based on the relevance of nodes, and section 3.3 discusses methods for presenting the results to the user.

The aim is to construct a reduced representation for a given input specification. Two methods are proposed to achieve this construction, both of which are guided by a criterion function that reflects the requirements of the user. The reduced graph is then presented to the user, who may alter their requirements or request a different level of detail in response. The input specification is hereafter referred to as the original graph. This process is called the visualization process and is illustrated in Figure 4.

The visualization process outlined in this paper is:

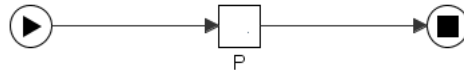
- 1: Calculate the relevance of each node according to the criteria
- 2: Reduce the graph by either the collapse or decimation methods
- 3: Display the graph to the user for inspection

The original graph is a graph  $G(V, E)$ , where each node  $v \in V$  is either a condition or a task. There is always one *start* condition,  $s$ , and one *end* condition,  $t$ . For the purposes of this paper, a *completed* graph has at least one task and every node  $v$  can be reached on a directed path from  $s$  to  $t$ . In other words, if  $p(v_i, v_j) \subset E$  returns the edges on a directed path from  $v_i$  to  $v_j$ ,  $\forall v \in V$ :



**Fig. 4.** Visualization support using the reduced graph approach

$p(s, v) \neq \emptyset \wedge p(v, t) \neq \emptyset$ . Any valid workflow specification, which must be a completed graph, can be abstracted to the level  $start \rightarrow P \rightarrow end$  (see Figure 5), which is the minimum valid specification possible.



**Fig. 5.** Any valid process model can be abstracted to the level of a single task  $P$ , which stands for “execute the process”

During the modelling process, where the user is still building the process specification, the graph may not necessarily be completed. Both of the graph reduction strategies proposed here support these partial graphs, however, the criterion function requires additional care to ensure that it also supports partial graphs. Partial graphs are those graphs where not all nodes can be reached from the start condition, or the end condition is not reachable from any node, or both. In practice the end condition is typically unreachable in a partial graph.

The aim is to build a reduced graph  $G_R(V_R, E_R)$  for an original graph  $G(V, E)$ , such that  $V_R \subset V$ . A relevance factor,  $\epsilon$ , is calculated by  $\epsilon_i = C(v_i)$  for each node  $v_i \in V$ , where  $C$  is the criterion function  $C : V \rightarrow \mathbb{R}$ .  $C$  orders the nodes according to their relevance to the task of the user.

### 3.1 The Criterion Function

The criterion function is formulated according to the task of the user. For example, if the task of the user is to identify deadlocks, then neighbourhood nodes that contribute to the deadlock state are of greater interest to the user than

the overall graph structure. Contrast this with a user that wishes to see only those nodes that contain a particular search term and closely related nodes. Consequently we assign each task a different criterion function, whose effects dictate the degree to which the preservation of structure overrides the relevance of neighbouring nodes.

**Structural importance** Preservation of the overall structure of the graph is achieved through identifying important control flow nodes. The control perspective defines the flow of control through the graph.

A promising heuristic structural importance measure is based on the connectedness,  $\chi : V \rightarrow \mathbb{Z}$ , of the node and its estimated position in the routing hierarchy,  $\phi : V \rightarrow \mathbb{Z}$ .  $\phi$  returns an integer calculated by counting the number of splits and subtracting the number of joins on the *shortest* path from  $s$  to the node, excluding this node.  $\chi$  is simply the sum of all connected nodes to this node.  $\epsilon$  is calculated as follows:

$$\epsilon_i = \frac{\chi(v_i)}{\min(\phi(v_i), 1)}$$

An example application of the heuristic structural importance criteria is shown in Figure 8.

**Text Retrieval** Text retrieval algorithms perform best when there are a number of words in a document. Business process models rarely include much text for each node, limiting the applicability of traditional text retrieval ranking methods. However, the context for a node can be viewed as the neighbouring nodes.

One approach to take advantage of this neighbourhood is to introduce a notion of *relevance flow*, which increases the relevance of nearby nodes. The amount of the contribution drops off with distance travelled including loops. The amount of the drop off is arbitrary and a constant rate,  $\beta$ , gives adequate results. The algorithm for graph  $G(V, E)$  is as follows:

- 1: Find  $S_T$ , the set of all nodes that contain the search term.
- 2: For each  $v \in S_T$ ,
- 3:     Initialise the contribution value,  $c \leftarrow 1$ .
- 4:     Initialise the neighbourhood node set,  $S_N \leftarrow \{v\}$ .
- 5:     While  $c > 0$  and  $S_N \neq \emptyset$ ,
- 6:         update  $\epsilon$  for all neighbours:  $\epsilon'(n) \leftarrow \epsilon(n) + c$  for all  $n \in S_N$ .
- 7:         reduce future contributions:  $c' \leftarrow c - \beta$ .
- 8:         update neighbour list:  $S_N \leftarrow \{n \in S_N : w \in V, \{nw\} \in E\}$ .
- 9:     End while.
- 10: End for.

An example application of the text retrieval criteria is shown in Figure 9.

**Graphical considerations** The business process model is a graphical representation, meaning that the modeller has assigned the positions of the nodes. These positions hold meaning, for example, invoicing related tasks will commonly be grouped together spatially. This meaning can be included in the criterion function by measuring the relative change in the position of a node.

### 3.2 Business Process Model Reduction

This section describes model transform techniques that produce reduced models based on the criterion function.

The reduced graph must preserve the semantics of the original graph to avoid being misleading. Semantics are preserved if all *possible orders of execution* of the remaining nodes are unchanged from the original graph. In other words, the dependencies between nodes cannot change.

Two methods are described: the *collapse* method, which incrementally reduces the graph until a threshold value for  $\epsilon$  is reached, and the *decimation* method, which removes all nodes below a threshold value and reconstructs the paths between remaining nodes.

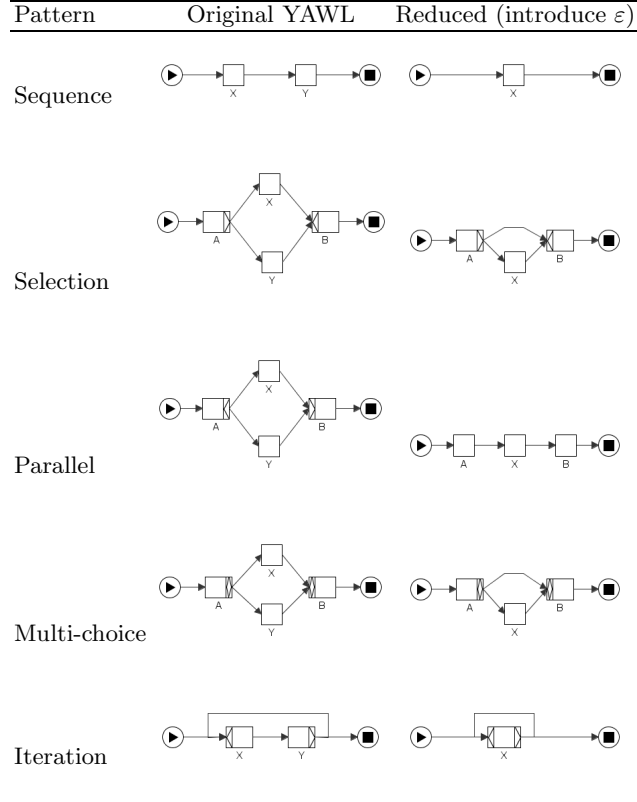
The threshold value is assigned by the user and is called the alpha-cut value, denoted  $\alpha$ .

**Collapse** The principle behind the collapse technique is to incrementally reduce the graph. Each incremental change in the graph is selected on the basis of removing the least relevant (minimum  $\epsilon$ ) node from the current model  $G_R^n$  to produce next  $G_R^{n+1}$ , according to conditions described next.

A non-join node is selected for removal at each increment. A split node is only selected if its predecessor is a task. The removal is performed by merging the node with its predecessor. Figure 6 illustrates how this is done under various circumstances. Split and join decorators are removed from a node when a single inflow or outflow, respectively, results from the collapse, yielding a sequence operation. Given the selection pattern under the heading ‘Original YAWL’ in Figure 6, the first selected node is  $y$ , which is merged with  $a$  to produce the version shown under the heading ‘Reduced (introduce  $\epsilon$ )’. Subsequently,  $x$  is chosen and merged with  $a$  to produce the sequence pattern of  $a \rightarrow b$ .

One advantage of the collapse technique is that the order of collapses can be stored. The inverse operation of a collapse, called a node-split, can then be performed to restore  $G_R^{n+1}$  to  $G_R^n$ . Another advantage is that since collapses relate one level of detail to another, the presentation can animate changes to increase interpretability of the technique. The calculation of collapses can be performed in a pre-processing step and since the actual collapse operation requires minimal processing, the visualization system can allow interactive navigation between various levels of detail.

An example application of the collapse algorithm is shown in Figure 8.



**Fig. 6.** Selected reduction patterns for the collapse technique

**Decimation** The decimation approach selects a number of nodes that will be included in  $G_R$ . All other nodes are *removed*. The original graph is then analysed to reconstruct the paths between the remaining nodes. Nodes are selected for inclusion if their relevance is  $\alpha$  or higher. A *concurrent* path is defined as any path from one node to another where a split exists on the path that was *not* synchronised before reaching the destination node. A *direct* path from  $x \in S$  to  $y \in S$  is a path from  $x$  to  $y$  without going through any other element of  $S$ .

The decimation-construction algorithm is given as follows:

- 1: Initialise the set of included nodes,  $S_I \leftarrow \{s, t\}$
- 2: add all  $v_i$  where  $C(v_i) > \alpha$  to  $S_I$ .
- 3: Initialise output edges,  $E_R \leftarrow \emptyset$
- 4: For  $x \in S_I, y \in S_I, x \neq y$ ,
- 5:  $V'_R \leftarrow V_R \cup \{y\}$
- 6: if there is a direct path from  $y$  to  $y$ ,  $E'_R \leftarrow E_R \cup \{yy\}$
- 7: if there is a direct path from  $x$  to  $y$ ,  $E'_R \leftarrow E_R \cup \{xy\}$
- 8: if a concurrent path  $\{x..y\}$  includes any  $z \in S_I$  ( $z \neq x \neq y$ ),

add the offending split node(s) before  $x$  and  $y$  to  $S_I$ ,  
 add the matching join node(s) after  $x$  and  $y$  to  $S_I$ .

9: End for

In practice, the detection of concurrent paths in step 8 is implemented by performing a search for elements of  $S_I - \{x\}$  in the original graph  $G$  starting at  $x$  and following the directions of edges in  $E$ . Split nodes will spawn multiple paths leading away from them, whereas join nodes reunite paths back together. Since the graph is cyclic, the search must keep a set of traversed edges to ensure they are not pursued again. Any path is terminated if it reaches  $y$ , or there are no untraversed outflows to follow. If any path finds an element of  $S_I - \{x, y\}$ , the path is recorded then terminated. The search is terminated when all paths terminate. The offending split nodes are found by backtracking recorded paths until a forward path to  $y$  is found. The matching join nodes are found using a similar approach that starts a path at every element of  $S_I - \{x\}$  that was previously found, including  $y$ . The algorithm continues until all paths terminate or there is a single path. It records join nodes that combine the paths along the way.

An example application of the decimation algorithm is shown in Figure 9 where the task ‘Negotiate on claim’ is an offending split node and ‘Complete settlement documentation’ is the matching join node.

### 3.3 Presentation Techniques

This section addresses the presentation of the reduced graph to the user. The reduction techniques described previously produce a reduced graph, but do not adjust the position of the nodes. The role of the presentation algorithm is to produce a visually pleasing layout of the reduced graph while preserving the intuitiveness of the result.

The original position of the nodes was chosen by the modeller and holds associated meaning. Whenever automated changes to the graph are performed, it is necessary to preserve the user’s mental map [17]. The relative position of nodes to one another should not alter considerably, since this would also reduce the ability of users to relate the reduced graph to the original graph.

The method used in figures 8 and 9 models the edges in  $E_R$  as springs that seek to achieve unit length. Nodes also carry a localised repellent force that pushes nodes apart and keeps them from overlapping. The system is then allowed to stabilise, which will occur when all edges have contracted until the repellent force of each node equals the spring force on its edges. This stabilisation can be animated, which adds to the visually intuitive nature of the reduction process. This animation is improved if the visualization is scaled such that the extents of the graph fit on screen.

To improve fidelity to the original graph another force can be modelled as an invisible spring between the node and its relative position in the original graph, called the scaled original position. The scaled original position  $P_s(i)$  of node  $i$  is

the position in the original graph  $P_o(i)$  scaled by the ratio of the extents of the graphs  $\gamma$ :

$$P_s(i) = \gamma P_o(i)$$

where  $\gamma$  is given by the ratio of the extents of the current graph  $E_c$  to the extents of the original graph  $E_o$ :

$$\gamma = \frac{E_c}{E_o}$$

## 4 Conclusions and Future Work

This paper proposed the use of a visualization process to support the understanding of large business process specifications. The process was divided into three steps that together provide the user with a simplified specification that is relevant to the task of the user.

The visualization process can be used to allow users to browse the specification in a similar manner to the way in which the web is explored through web search engines. The techniques presented in section 3 work with partial graphs, allowing the process to be used during modelling of the business process specification.

The notion of a criterion function allows for rich and flexible expression of relevance. Section 3.1 provided two criterion functions: one favouring the overall structure of the graph and another for locating nodes related to a text search term. Section 3.2 described two methods for actually building the reduced graphs: the *collapse* technique, which is incremental, and the *decimation* technique. Section 3.3 covered aesthetic and interpretability issues involved with presenting the reduced graph to the user. Two examples were given in figures 8 and 9 for the original graph in figure 7.

Future work would explore additional criteria functions. User interaction with the system can also be extended, such as to allow users to ‘brush’ over nodes to reveal their neighbourhood. The applicability of this visualization process is not limited to modelling large processes. It can be extended to work with run-time data or process mining results.

**Acknowledgements** The authors wish to acknowledge Arthur ter Hofstede for his feedback and Michael Roseman for providing access to real world large business process specifications. Thanks also go to Alexander Campbell, Rune Rasmussen, and Frederic Maire for their suggestions on graph theory.

## References

1. W. van der Aalst, “Business process management demystified: A tutorial on models, systems and standards for workflow management,” in *Lectures on Concurrency and Petri Nets*, vol. 3098, pp. 1–65, Springer Verlag, Berlin, 2004.
2. M. M. Burnett, M. J. Baker, C. Bohus, P. Carlson, S. Yang, and P. van Zee, “Scaling up visual programming languages,” *Computer*, vol. 28, no. 3, pp. 45–54, 1995.

3. J. M. Mendel, *Uncertain Rule-Based Fuzzy Logic Systems*. Prentice Hall PTR, 2001.
4. D. Luebke, M. Reddy, J. D. Cohen, A. Varshney, B. Watson, and R. Huebner, *Level of Detail for 3D Graphics*. Morgan Kaufman Publishers, 2003.
5. H. Hoppe, "Progressive meshes," in *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, (New York, NY, USA), pp. 99–108, ACM Press, 1996.
6. W. van der Aalst and K. van Hee, *Workflow Management: Models, Methods, and Systems*. MIT Press, 2002.
7. W. van der Aalst and A. ter Hofstede, "Yawl: Yet another workflow language," in *Information Systems*, vol. 30, June 2005.
8. W. van der Aalst, A. ter Hofstede, B. Kiepuszewski, and A. P. Barros, "Workflow patterns," *Distrib. Parallel Databases*, vol. 14, no. 1, pp. 5–51, 2003.
9. P. Keller and M. Keller, *Visual Cues*. IEEE Press, 1992.
10. S. Henderson, "Vised: Visualization techniques." Retrieved 25 June 2004 from <http://www.siggraph.org/education/materials/HyperVis/vised/VisTech/vtmain.html>, 1996.
11. M. Reed and D. Heller, "Olive: Online library of information visualization environments." Retrieved 15 May 2004 from <http://www.otal.umd.edu/Olive/>, 1997.
12. S. Card and J. Mackinlay, "The structure of the information visualization design space," in *IEEE Symposium on Information Visualization*, pp. 92–99, IEEE Press, Oct 1997.
13. E. Chi, "A taxonomy of visualization techniques using the data state reference model," in *IEEE Symposium on Information Visualization*, pp. 69–75, IEEE Press, Oct 2000.
14. K.-L. Ma, "Visualization - a quickly emerging field," *ACM Computer Graphics*, vol. February, pp. 4–7, 2004.
15. R. Brown and B. Pham, "Visualisation of fuzzy decision support information: A case study," in *IEEE International Conference on Fuzzy Systems*, 2003.
16. L. J. Campbell, T. A. Halpin, and H. A. Proper, "Conceptual schemas with abstractions: Making flat conceptual schemas more comprehensible.," *Data Knowl. Eng.*, vol. 20, no. 1, pp. 39–85, 1996.
17. K. Misue, P. Eades, W. Lai, and K. Sugiyama, "Layout adjustment and the mental map," *Visual Languages and Computing*, vol. 6, no. 2, pp. 183–210, 1995.

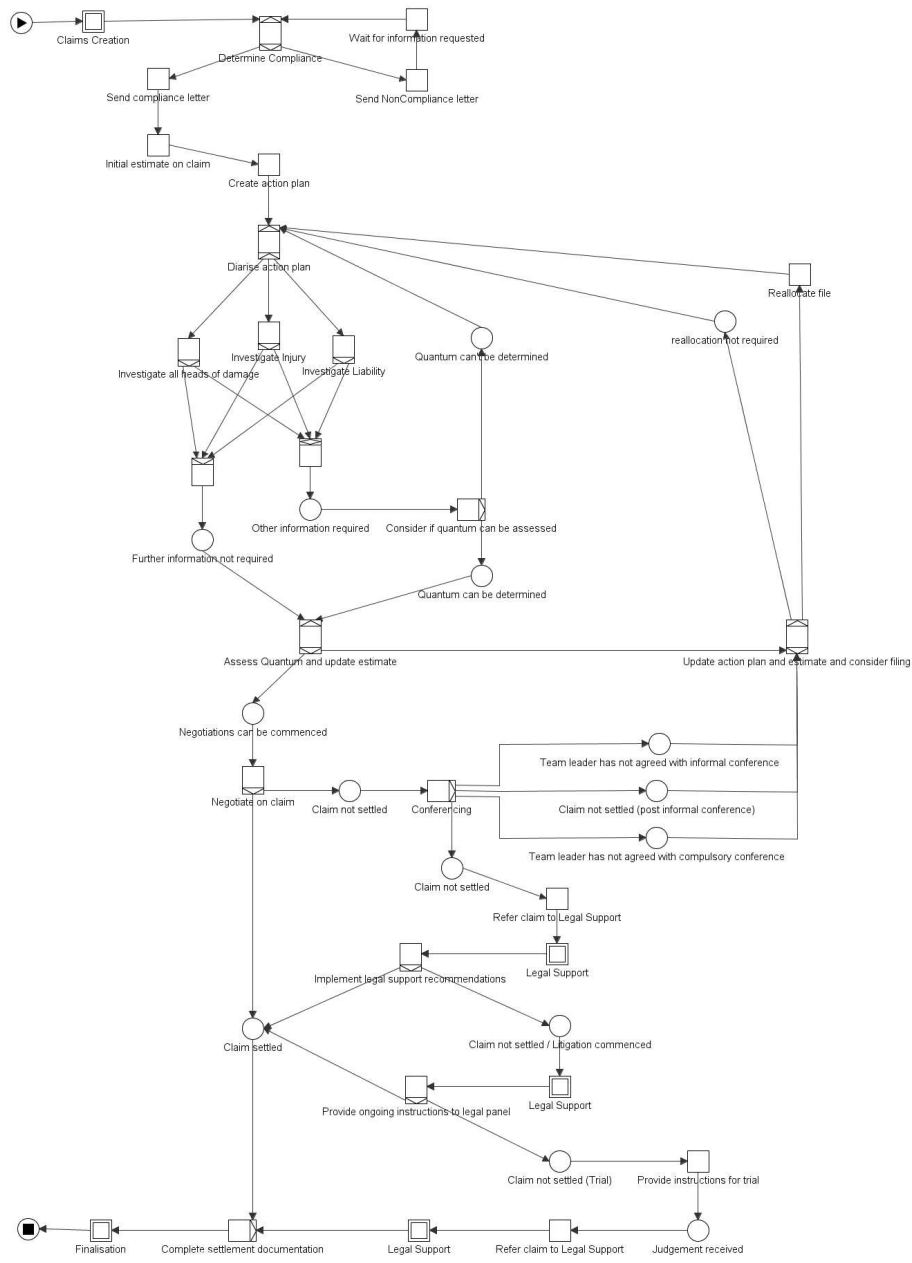
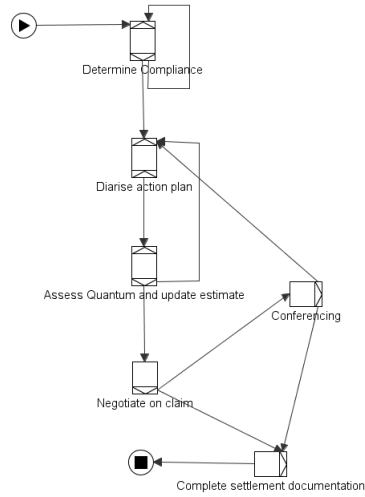
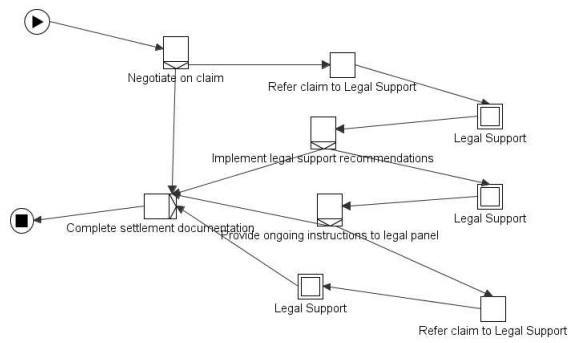


Fig. 7. An example workflow specification that is too large to fit on a computer screen.



**Fig. 8.** A reduced version of specification graph in Figure 7 showing structure built using the collapse approach ( $\alpha = 2.5$ ).



**Fig. 9.** A reduced version of the specification in Figure 7 for the text query 'legal' built using the decimation approach ( $\beta = 0.5$ ,  $\alpha = 1$ ).