Improvise: Interactive Multimedia Process Visualization Environment

Naser S. Barghouti, Eleftherios Koutsofios and Edith Cohen

AT&T Bell Laboratories, 600 Mountain Avenue, Murray Hill, NJ 07974, USA
{naser, ek, edith}@research.att.com

Abstract. Improvise is a multimedia system for modeling, visualizing and documenting software and business processes. It runs under Microsoft Windows and on most flavors of the UNIX operating system. Improvise provides facilities for drawing multi-layered process flow diagrams and graphical information models, and annotating the nodes and edges of the diagrams with multimedia information and executable attachments. The set of node shapes used to create process flow diagrams and information models is customizable. Each node and edge in the diagram is treated as an object with a customizable set of attributes that can be assigned values interactively. The values of some of the attributes are used to compute expected cost and time of a process. Users can browse the diagrams, zoom in on specific parts of the process, display the attached multimedia information, and execute the systems used in performing the process. Improvise is an open system that can be easily connected to other tools, such as process simulation and performance analysis tools. It has already been integrated with the process support environment Marvel as part of the implementation of Provence.

1 Introduction

Software process models describe how software products are developed, distributed and maintained. In the past few years, several process modeling formalisms and process support systems have been developed (for example, Merlin [10], Adele [4], EPOS [5], SPADE [1], Oikos [15], Arcadia [11], HFSP [12], Process Weaver [6], and Marvel [2]). In spite of the expressive power and elegance of many of these formalisms and systems, they have not, by and large, made their way into real use by corporations. It may be argued that most of these systems are research prototypes that were never intended to be used in industry. However, since several of these systems are being used as a basis for developing products, it is useful to investigate whether or not there are obstacles that would limit their use, once they have become products.

From our experience, one major obstacle is technology transfer as manifested in two problems. First, many organizations are not ready for these formalisms and systems because using them requires certain programming and abstraction skills that are lacking in process engineers, making the learning curve too steep. Thus, even though the technologies supported by these systems might be very useful to corporate organizations, the sophistication of the systems, coupled with the lack of support for incremental and layered introduction of the technologies, becomes a limiting factor. Second, most of the systems and formalisms were developed in research computing environments that are quite different from the computing environment of industrial organizations: Whereas a typical research computing environment is composed mostly of workstations running some flavor of the UNIX operating system, the computing environment of many industrial organizations is composed of a combination of UNIX workstations and personal computers (PCs) running either Microsoft Windows or the Macintosh operating system.

The technology transfer problem is thus a result of the discrepancy between the skills and the computing environment in corporate organizations on one hand and existing process support systems on the other. There have been a few recent efforts to address aspects of this problem. For example, the process support system Leu advocates a unified and incremental approach to software and business process modeling [9]; another system, Teamware, also addresses some aspects of the discrepancy [17]. Yet, the discrepancy remains largely unbridged.

Improvise is a tool that aims to bridge this discrepancy in three ways. First, it espouses a mostly graphical means for modeling software and business processes at multiple levels of detail. Second, it runs under Microsoft Windows as well as under most flavors of the UNIX operating system. Third, it is based on an open architecture, in the sense that it provides a programmable interface for interoperating with the other tools that comprise the computing environment of an organization, such as word processors, project management tools, process simulation tools, and so on.

Improvise integrates object-oriented process modeling with multimedia technology to provide a process modeling environment that is powerful in its simplicity and ease of use. The basic idea is to allow a process engineer to create graphically a multi-layered process flow diagram and an information model, and to annotate the diagrams with multimedia attachments (e.g., video, images, text) and performance estimation information. Improvise automates the layout of the diagrams, the computation of the estimated cost and time of the process based on the data annotations, the translation of the graphical process model into other process modeling formalisms, and interaction with other tools.

To demonstrate the utility and expressive power of Improvise, we have used it to model several corporate processes within AT&T. We have also distributed it to process engineers to evaluate its ease of use. Initial results are very promising and indicate that there is a very large pool of users who can benefit from it. The main advantages seem to be its very simple graphical interface, its support for incrementality, its incorporation of multimedia capabilities, and its portability between the UNIX and Microsoft Windows operating systems

The rest of the paper is organized as follows. We first explain the motivation behind Improvise and give a brief history of the project. Next, we present a high-level description of the system, followed by a more detailed discussion of each of its facilities. We then explain the mathematical model used to compute useful process properties. Next, we present a case study in modeling a service provisioning process. Based on our experience, we examine the limitations of the current implementation of Improvise, and we discuss future directions of the work. Finally, we conclude with a summary of the paper.

2 Background and Motivation

The concept behind Improvise grew out of Provence, an open architecture for process modeling, enactment, monitoring and visualization [14]. The role of the visualization component in the original Provence architecture was *output visualization*: Whenever a transition in the process model enactment was performed by the process server, the visualizer was called to update the visual representation of the process model enactment to reflect the results of the transition. Thus, the visualization component was reactive, which has proven inadequate in practice.

In implementing Provence, Marvel [2] was used as the process server. Marvel provides a powerful process modeling language that combines objects and rules, and that supports tool integration through an enveloping mechanism. However, after using Marvel to model several corporate processes (see, for example, [3]), it became apparent that there was a need for a system that provided higher-level and simpler modeling primitives. This realization was reinforced during several interactions with process engineers in the organization within AT&T responsible for provisioning customer services on the AT&T network.

Process engineers in this organization had already created numerous process flow diagrams which they wanted to use as the basis for more detailed process modeling and analysis. Using Marvel or a similar system would have forced them to learn a new process modeling formalism and start from scratch in their modeling efforts. Furthermore, most process engineers in the organization use PCs running Microsoft Windows, which Marvel does not run under. What was needed then is a system that runs on PCs, accepts input that is the same or similar to the process flow diagrams already created by the engineers, and uses these diagrams as a basis for more detailed modeling using other process analysis and simulation systems, such as Marvel.

To address this need we enhanced the visualization component of Provence, in a relatively short period of time, to provide *input visualization* in addition to output visualization. The term *input visualization* is used here to mean a visual and interactive method of creating a process model. Such a component enables process engineers to use a simple graphical notation to create models of their processes. Later, the goals of the project were expanded to include three capabilities: (1) annotating the graphical process models with multimedia attachments (e.g., a video clip), cost and time data; (2) performing various kinds of high-level process analysis, such as estimating the expect cost of executing a process; and (3) translating the graphical process models into other formalisms, such as Marvel's rule-based formalism.

3 Process Modeling with Improvise

In general, a process model comprises two components: an activity model and an information model [3]. The activity model describes the process steps, the resources needed to carry out these steps, and the dependencies among the steps. The information model describes the artifacts produced by the process, the data consumed and produced during the execution of the process, and the relationships among the data. Improvise adopts a graphical representation of both components.

3.1 Process Flow Diagram

A process flow is one means of describing the activity model of a software process. A process flow depicts a set of connected steps which together achieve a desired goal, such as completing a product or providing a service. The information needed by one step may be supplied by another step in the same process, a different process, or an external source, such as a human designer or another organization. The performance of a step may involve the execution of one or more software tools.

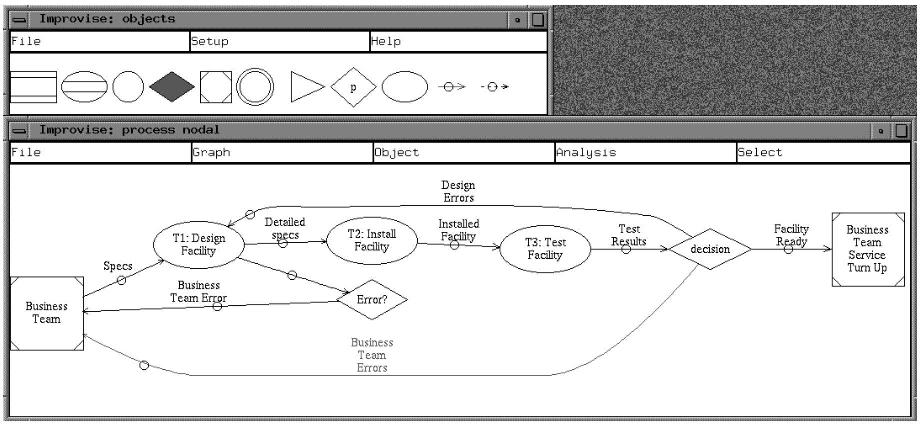

Fig. 1. Creating a Process Flow with Improvise.

A process flow is, thus, a set of relationships between objects that represent process steps, human developers, software tools, organizations, and so on. Improvise espouses a graphical model that represents a process flow as a multi-layered directed graph, which has proven to be an excellent means for presenting relationships between objects. Improvise displays a menu of available nodes and edges, and provides a drawing window in which the user can create a process flow diagram.

The top window in Figure 1, labeled `Improvise: objects`, is the menu of currently-available node types. The menu contains a default (but customizable) set of node shapes representing different parts of a process flow diagram, and two kinds of directed edges to model dependency among these parts. The default set of node types and edge types, as shown from left to right in the window, is:

1. *hamburger-shaped box* to represent a manual task,
2. *ellipse with two lines* to represent a computer-assisted task.
3. *circle* to represent a system,
4. *diamond* for a decision point,
5. *square with lined corners* for external source/destination of data,
6. *dual circles* as connection points to other processes.
7. *triangle* for feedback connectors,
8. *diamond with "P" in it* to represent an external process,
9. *ellipse* to represent an abstract process,
10. *solid arrow* to represent input/output relationship, and
11. *dashed arrow* to represent other kinds of relationship between objects.

Node types can be added/removed to/from the menu via the procedural language provided by Improvise. The small circles in the middle of the edges are for grabbing the edge by the mouse device. The user selects a node or edge type from the menu by clicking the left button of the mouse devise on the appropriate shape; the selected node type and edge type are highlighted. In the figure, the decision node type has been selected from the menu.

After selecting a node type from the menu, the user proceeds to create an instance of the selected node type in the active drawing and browsing window, which is the larger window in Figure 1, labeled `Improvise: process nodal`. Clicking the left mouse button inside a node and dragging it to a second node creates a directed edge from the first node to the second. In Figure 1, a process flow has been created with three subprocesses, two decision point, one external source of data, and one external destination of data.

The flow in Figure 1 describes a simple service provisioning process: A business team provides a set of specifications for a new network facility to the design team. The design team inspects these specification; if it detects any errors, it sends the specifications back to the business team. Otherwise, the required facility is designed (subprocess `T1`). Once the facility is designed, a set of design specs are made available to the installation team, which proceeds to install the facility (subprocess `T2`). After the facility has been installed, it is tested by the testing team (subprocess `T3`). Design errors detected during testing are communicated back to the design team, while business team errors are communicated back to the business team. Once the facility has been installed and tested successfully, it is sent to the service turn up team, which makes the facility available to customers.

3.2 Multi-Layer Capability

Each of the three subprocesses of the process depicted in Figure 1 can be described in more detail in a second layer. For example, Figure 2 shows a second

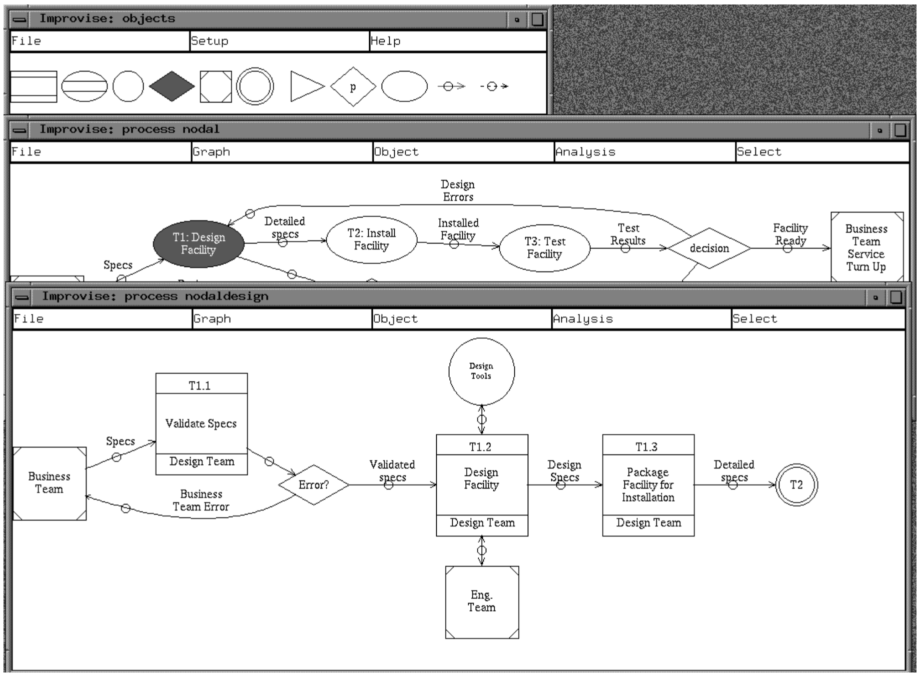

Fig. 2. A Multi-Layered Process Flow Diagram.

layer, in the window labeled `Improvise: process nodaldesign`, detailing the subprocess of designing a network facility. The first task (labeled `T1.1`), which is a manual task, validates the input specs from the business team, communicating back any errors to the business team. The second task (labeled `T1.2`) is a computer-assisted task since it involves the use of some design tools, as shown in the figure. The task also requires assistance from external experts (the box labeled `Eng. Team`). The last task in the subprocess (labeled `T1.3`) is another manual task whose output is the detailed design specs that are used to install the facility. The connection between this subprocess and another subprocess at the higher level is modeled explicitly through the connector node labeled `T2`.

3.3 Automatic Layout of Flow

For a graph drawing to be useful in understanding the relationships between objects, the layout of the drawing must be clear and easy to read. For example, the diagram in Figure 1 shows clearly the sequencing of the three tasks, the input/output relationship among them and various other relationships with systems and other processes. Improvise automatically layouts the diagrams, using a set of aesthetic criteria, such as avoiding edge crossings and edges crossing over nodes whenever possible. Thus, the user need not worry about the placement of nodes in a process flow diagram because Improvise will automatically, on-demand, lay out the diagram to make it as nice as it can. Editing a diagram is also made simpler because of this automatic layout facility.

3.4 Multimedia Capabilities

Improvise treats the nodes and edges in a process flow diagram as objects, each with a pre-defined (but customizable) set of attributes; the particular set of attributes depends on the type of node. The user can select a node or an edge object by clicking on it with the left button of the mouse devise; the selected object is highlighted in red. Selecting an object opens an editing window, where the user can view and change the values of the attributes of the object. The values of some of the attributes (e.g., the name, description, and organization of a task node) appear in the process flow diagram, while others are stored but are not visible.

Some of the attributes of node types are designated as multimedia attributes. The value of such an attribute can be a video or audio clip, a graphical image, or a textual image. For example, the attribute `video` of the task node `T1.1` has the value "v1", as shown in the window labeled `Improvise: object attributes` in figure 3. This means that there is a video clip associated with this node, stored in the file whose name is "v1". This video clip can be shown by selecting `show video` from the menu. The multimedia clips are shown using commercial tools that Improvise starts up with the clip name as argument. This allowed Improvise to be ported to many different environments and helped us work around the problem that there are no standards for video storage and playback; each vendor supplies its own version.

In Figure 3, the video clip attached to the task node labeled `T1.1` is being played. The user can stop the video at any time, rewind it, forward it or pause it. Audio clips can be played on the machine in a similar way. The video features a member of the design team explaining how he performs the task of validating the specifications he receives from the business team. In the two-minute video, the designer shows how he inputs the specification in the database, inspects each entry, and validates the entries.

The role of multimedia attachments in process modeling has not fully been investigated yet. From our limited experience, it is clear to us that images, videos and audio clips can play a very important part in clarifying the various steps in a process document. More important, they can be used quite effectively for

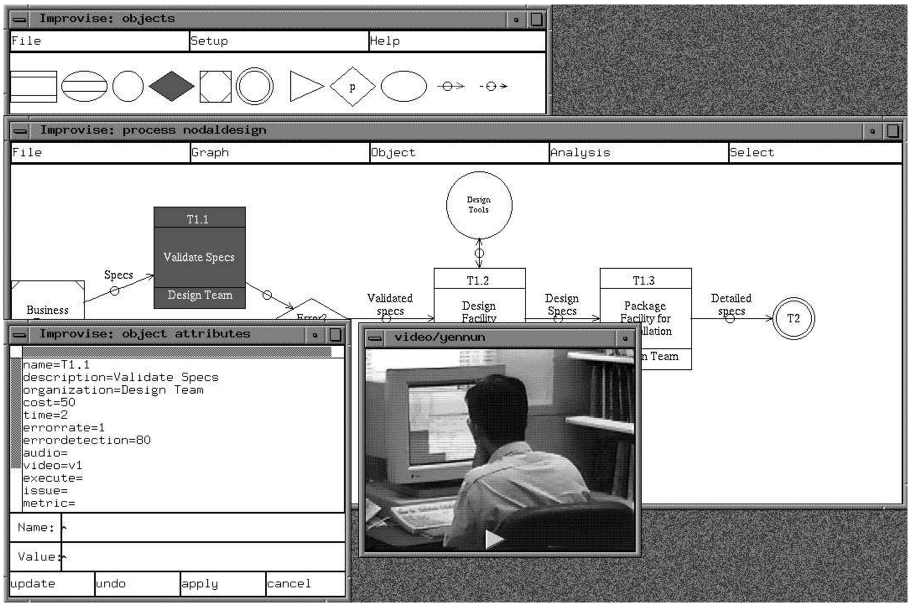

Fig. 3. Multimedia Capability in Improvise.

training new members of a team. In addition, videos tend to raise the awareness of process engineers as to the human factor in executing the process. Actually watching a developer or a designer carry out a task might give the process engineer, who might be in a remote location, as is often the case in AT&T, a better feel for how to use available human resources to improve the efficiency of the process.

3.5 Graphical Information Model

Improvise supports graphical, object-oriented information modeling. It provides a graphical user interface for defining object classes, inheritance hierarchies, and relations between instances of the classes. The classes describe the entities that participate in a process, such as humans, material resources, organizations, and so on, and the types of information produced by and accessed during the execution of a process, such as structured documents, source code, data repositories, and so on.

Figure 4 shows two windows. The top window, labeled `Improvise: objects` is the information modeling menu of objects and relationships. It shows one kind of node that represents a class, and five kinds of edge representing different kinds of relationships. These relationships, from left to right, are:

- containment of instance of one class in an instance of another,

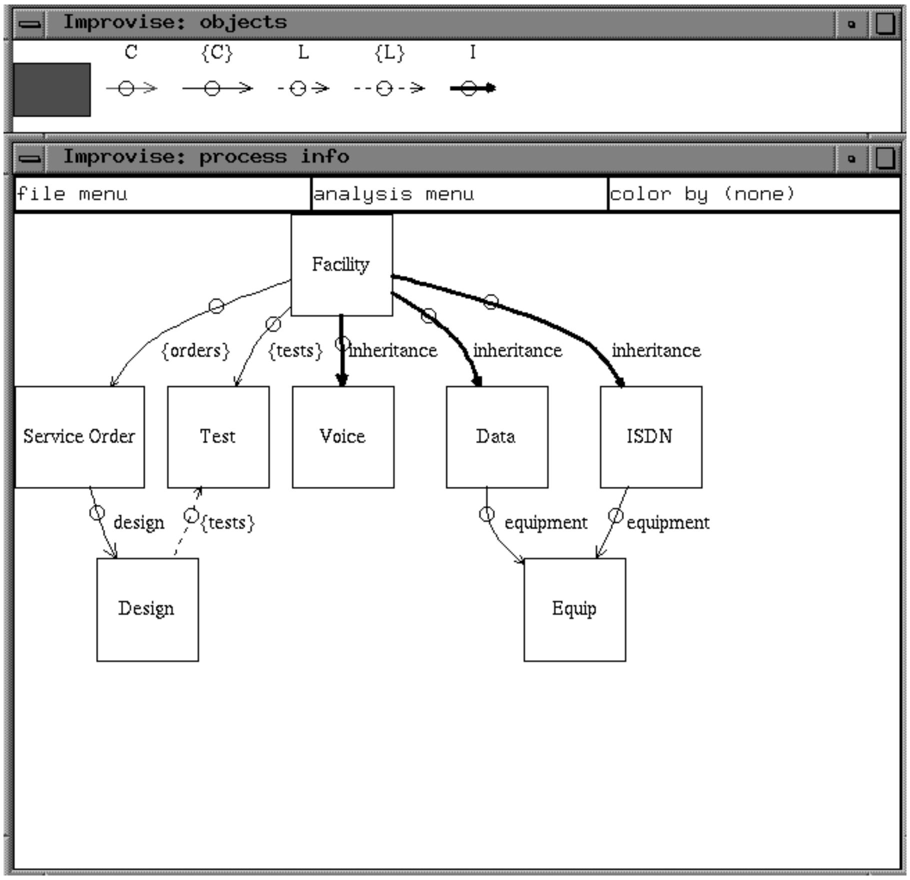

Fig. 4. Creating an Information Model with Improvise.

- containment of a set of instances of one class in an instance of another class,
- link (arbitrary relationship) between an instance of one class and an instance of another class,
- a set of links between an instance of one class and a set of instances of another class, and
- inheritance relationship.

The bottom window depicts an example information model that describes the types of information accessed by the service provisioning process discussed in Section 3.1. The information model comprises seven classes with various relationships between them. For example, the diagram shows that the three classes `Voice`, `Data` and `ISDN` are subclasses of (inherit from) the class `Facility`. It also shows that an instance of the class `Facility` may contain a set of instances of the class `Service Order`.

4 Mathematical Model for Process Analysis

One of the most important reasons for modeling a process formally is to be able to analyze it by characterizing some of its properties. The purpose of the analysis is to detect problems in the process and to explore areas for improvement. Typical problems that are detected during analysis include input/output discrepancies between process steps, inefficiencies in scheduling, and incorrect sequencing. There are several commercially-available tools that assist in various kinds of process simulation and analysis, using discrete event simulation and queueing theory. In addition to being expensive, most of the commercially-available tools require a fairly detailed process model.

Our experience within AT&T, however, is that very useful process analysis can be performed on simple process models like the ones we have described thus far in the paper. In particular, process engineers can explore various potential improvements to a process by modifying the performance data in a process model and comparing the performance of the altered process with the original one.

The three measurements used most frequently to compare process performance are the expected cost of a process (which includes money, resources, etc.), the expected execution time of a process, and the percentage of re-work due to errors. Improvise provides a mechanism to automatically compute these three properties, given that nodes in the process flow diagram are annotated with individual cost, time and error rate data. The computation is performed on demand after the user has selected a subset of the process to analyze. The subset is selected by determining a starting node and one or more termination nodes; the starting and termination points must be in the same layer (see Section 3.2). The user can change the data associated with nodes and re-run the computation to perform "what-if" scenarios.

4.1 The Model

As explained above, the set of node types and the attributes associated with each node type are customizable in Improvise. Therefore, the process analysis model must abstracts away from the specific set of node and edge types in a particular process model. After careful examination, we have determined that for the purpose of analysis there are four abstract categories of nodes in any process flow: *normal nodes*, *error sources*, *decision nodes*, and *null nodes*.

Normal nodes contribute to the expected cost or execution time of a process. The *error sources* category is the subset of *normal nodes* that contains those normal nodes that generate erroneous output, and thus lead to some degree of re-work. *Decision nodes* have two or more edges coming out of them, but only one of those edges is traversed in a single execution of the decision node. There are two kinds of edge going out of a decision node: forward edges to subsequent nodes and backward edges to previous nodes (e.g., going back to an error source after detecting an error). *Null nodes* contain process-related information, but they do not affect the performance of the process because they do not contribute to the generation of errors, time, or cost of the process.

To perform simple analysis, the process model must distinguish between these four categories of nodes, and it must contain data that can be used for the analysis. Improvise distinguishes among the four node types as follows. Error sources are distinguished by having an attribute called `error rate` with a non-zero value; the value of this attribute is the percentage of erroneous output generated by the node. Several types of nodes may be designated as potential sources of error. Normal nodes must have an attribute `cost` with a non-zero value, an attribute `time` with a non-zero value, or both. Decision nodes are represented in Improvise by a diamond, as described in Section 3.1. The nodes that do not fit into any of these three categories are considered null nodes.

In addition to distinguishing between different types of nodes the analysis model requires that each of the edges coming out of a decision node (both the forward and back edges) have a probability associated with it. The probabilities of all the edges coming out of a single decision node must add up to 1.

After categorizing the nodes in the selected subset of the process flow diagram, we generate a directed graph using only normal nodes (which include error source nodes). The directed graph consists of a set of nodes U and a set of edges E. We then use the probabilities on the edges coming out of decision nodes and the values of the attribute `error rate` of the nodes that are in the error sources category to build a *Markov Process* over the set of nodes U. The purpose is to compute the expected number of visits of each node in U, which when multiplied by the values of the attributes `cost` and `time` gives us the expected cost and time of a single execution of the process.

A *plain Markov model* over the set of nodes U implies that the probability distribution of a node is independent of the previous nodes in a chain of nodes, and thus cannot depend on the state of the node. This independence assumption is not always justified. For example, consider the service provisioning process discussed in Section 3.1, where the initial service order from the business team may contain an error that is not detected except after performing several steps. When the error is detected at some step, it is sent back to the source of the error to be corrected. For this process, which step is performed next depends not only on the current step but also on whether the service order being processed in the step is "correct" or "contains an error".

Our model takes into consideration the state of the nodes. We assume that each node in U can be in any one of C *internal states*. The internal states correspond to the number of error sources in the process model. In general, a process execution reaches a node u with either correct data, or with incorrect data due to any one of the n sources of error in the process. Therefore, if there are n nodes that are sources of error in the process flow diagram, then each node in U can potentially be in any one of $n+1$ internal states. The probability of traversing an edge $e = (u, u')$ between two nodes u and u' depends on the internal state c of node u and on the probability of the corresponding edge in the process flow diagram. Hence, our model is a *Markov Process* over a set of states $U \times C$.

For every edge $(u, u') \in E$ and every pair of internal states $\{c, c'\} \subset C$, we compute

$$p(u, c, u', c') \in [0, 1]$$

the probability that when the process execution reaches node u in internal state c, the next state in the execution will be node u' in internal state c'. We use the default assumption that when $(u, u') \notin E$ (i.e., there is no edge between u and u') then $p(u, c, u', c') = 0$.

The sum of the probabilities of traversing the edges going out of a node is 1. More formally, for all $u \in U$, $c \in C$ we have

$$\sum_{(u', c') \in U \times C} p(u, c, u', c') = 1 \ .$$

4.2 Computing Expected Number of Visits

For each node u, Improvise calculates $v : U \times C \to R_+$, where $v(u, c)$ is the expected number of visits of node u while at internal state c, and R_+ is the set of all positive real numbers. Calculating the expected number of visits allows us to obtain desired quantities, such as the expected overall cost or time of the process.

The calculation considers an expanded flow graph on the set of states $U \times C$. When the expanded graph is acyclic, the expected number of visits can be computed for each node by considering the nodes according to a topological ordering:

1. The start state (u, c) has $v(u, c) = 1$.
2. When we consider a node (u, c) with predecessors (u_i, c_i) $(i = 1, \ldots, k)$, we have
$$v(u, c) = \sum_{i=1}^{k} v(u_i, c_i) p(u_i, c_i, u, c) \ .$$

When the expanded graph contains cycles, the solution requires solving the following system of linear equations:

1. For all $(u, c) \in U \times C$,
$$v(u, c) = \sum_{(u', c') \in U \times C} v(u', c') p(u', c', u, c)$$
2. Let $T \subset U \times C$ be the set of terminal states.
$$\sum_{(u, c) \in T} v(u, c) = 1$$

4.3 Computing Other Desired Quantities

The expected number of visits enable us to obtain other properties of the process:

1. The expected number of times an edge (u_1, u_2) is traversed is:

$$\sum_{c \in C} v(u_1, c) \sum_{c' \in C} p(u_1, c, u_2, c') \ .$$

2. Suppose that the nodes and transitions have costs/staff-hours estimates $L(u, c)$ and $L(u, c, u', c')$. The expected cost incurred at a task node u is

$$\sum_{c \in C} v(u, c) L(u, c) \ .$$

The expected cost incurred at an edge (u_1, u_2) is

$$\sum_{c \in C, c' \in C} v(u, c, u', c') L(u, c, u', c') \ .$$

3. The expected total cost per execution of the process is

$$\sum_{(u,c) \in U \times C} v(u, c) L(u, c) + \sum_{(u,c,u',c') \in U \times C \times U \times C} v(u, c) p(u, c, u', c') L(u, c, u', c') \ .$$

5 Implementation, Limitations and Future Work

Improvise is implemented as an enhancement of *dotty*, a customizable, general-purpose graph editor that was used as the visualization component in the Provence architecture [16, 14]. *dotty* provides a library of functions for creating, editing, browsing and querying directed or undirected graphs. It can be used either as a stand-alone tool or as a graphical front-end for other applications. A procedural language is provided for customizing its graph editing facilities and for building interfaces between it and applications [13]. *dotty* has already been customized in that way to visualize various kinds of software engineering information, including data structures, database schemas, program call graphs, finite state machines, and the results of database queries [16].

The result of customizing *dotty* in the case of Improvise is a graphical front-end that displays a menu of available nodes and edges, and provides a drawing window in which the user can create a process flow diagram and an information model. *dotty* implements an automatic graph layout algorithm based on a set of aesthetic criteria, such as avoiding edge crossings and edges crossing over nodes whenever possible (and it is not always possible), and drawing edges as smooth curves [8]. These criteria help reduce layout artifacts that tend to distract the user, instead allowing the user to focus on the information and its relationships.

In addition to the graph editing capabilities, *dotty* has two other features that make it a good choice as a basis for implementing Improvise. First, the library of functions it provides is portable to both Microsoft Windows and most flavors of the UNIX operating system. Thus, we were able to port Improvise to both operating systems rather rapidly. Further, users can use both the UNIX version and the Microsoft Windows version interchangeably on the same process flow diagram and information model. This has proven to be a very important feature for the users of Improvise. Second, *dotty*'s programmable interface allowed us to integrate it with Marvel with relative ease. This integration, in addition to being useful, demonstrates the ease of integrating Improvise with other process simulation and analysis tools.

In spite of the very positive feedback we have got from the users of Improvise so far, there are a few limitations that have become apparent and that we are working to overcome. First, the information model diagram and process flow diagram of a single process model are handled separately. The current implementation does not provide any mechanisms for relating the two diagrams. Integrating the process flow diagram and the information model is required to answer queries about a process model of the following nature: "Which steps in the process change a specific piece of information stored in a database", "What are the data dependencies between two subprocesses", and so on.

Second, the mechanism provided by Improvise for customizing the node types and attributes used to model a process is cumbersome since it requires programming. It is possible to replace this programming interface with a graphical interface that provides a large set of possible icons and drawing primitives, allowing the process engineer to design the specific node shapes and edges.

Third, the mathematical model is limited in that it can compute only the expected total cost and staff-hours of a process execution, but not the elapsed time. To compute elapsed time we need to extend the mathematical model to handle parallel processing of process steps and to account for the history of the execution so far. For example, in the service provisioning example, it is unlikely that for a service order that contains a minor error to go through the exact same steps the second time around; more likely, some of the steps will be skipped and other might be performed faster because part of the previous processing might still be applicable.

Finally, the current information modeling facilities are limited. We would like to extend them to handle the full power of information modeling provided by systems like Marvel [2] and Leu [9]. These include the ability to visualize not just the class definitions but the instance hierarchy, the ability to model arbitrary relationships between objects, and support for evolution of the information model.

We are currently extending Improvise to overcome these limitations and to support multiple geographically-distributed engineers. A multiuser version of Improvise requires facilities for coordination, reliability, and replication of data. A first step in providing multiuser support is enhancing Improvise with a process repository. The problem is finding an appropriate repository that is portable across UNIX and Microsoft Windows operating systems.

6 Summary and Conclusions

The paper presented Improvise, a multimedia, graphical tool for process modeling, visualization and analysis. Improvise provides facilities for graphically defining both the information model and the activity model of a process. The activity model is defined in terms of a multi-layered process flow diagram, where the shapes of nodes used in the diagram, and the attributes associated with each node type are customizable. The information model is defined in terms of a class relationship diagram with two views, an inheritance hierarchy view and an object composition view.

Improvise uses the facilities of the underlying graph editor *dotty* to automatically lay out the diagrams according to several aesthetic criteria, such as avoiding edge crossings and edges crossing over nodes whenever possible, and drawing edges as smooth curves. Improvise also allows users to annotate the process flow diagram with multimedia attachments, such as text, graphics, audio and video. These multimedia attachments enhance the power of the process model and are particularly useful for training and process documentation.

To demonstrate the functionality of Improvise, we have modeled several corporate processes. These processes include the process used by AT&T to provision services to customers. The process diagrams in Figures 1, 2 and 3 are adapted from that process model. We are now investigating how to extend this model to cover other corporate processes, and how to use Improvise for process re-engineering. We are also exploring the use of Improvise for process documentation control, which requires connecting it with the World Wide Web. Another application we are investigating is training and re-training of process personnel.

We believe that three crucial factors contributed to the success of Improvise so far: (1) its portability across both Microsoft Windows and most flavors of the UNIX operating system, (2) its graphical input visualization approach, and (3) its ease of use. The openness of its implementation, which allows us to integrate it with more sophisticated process support, analysis and simulation systems, enables us to define a path toward introducing these systems into corporate organizations.

Acknowledgements

We thank Kaveh Hushyar and Shaikh Amin from the Network Services Division of AT&T for their support and valuable input throughout this project; they have been an invaluable source of information for us.

References

1. Sergio Bandinelli, Alfonso Fuggetta, Carlo Ghezzi, and Luigi Lavazza. SPADE: An Environment for Software Process Analysis, Design, and Enactment. In *[7]*, pages 223–248. 1994.

2. Naser S. Barghouti and Gail E. Kaiser. Scaling Up Rule-Based Development Environments. In *Proc. of 3rd European Software Engineering Conference, ESEC '91*, pages 380–395, Milan Italy, October 1991. Springer-Verlag. Published as *Lecture Notes in Computer Science* no. 550.
3. Naser S. Barghouti and David S. Rosenblum. A Case Study in Modeling a Human-Intensive, Corporate Software Process. In *Proc. of the Third International Conference on the Software Process*, pages 99–110, Reston, VA, October 1994.
4. Noureddine Belkhatir, Jacky Estublier, and Welcelio Melo. ADELE-TEMPO:An Environment to Support Process Modelling and Enaction. In *[7]*, pages 187–221. 1994.
5. Conradi, Reidar *et al.* EPOS: Object-Oriented Cooperative Process Modelling. In *[7]*, pages 33–70. 1994.
6. Christer Fernström. PROCESS WEAVER: Adding Process Support to UNIX. In *Proc. of the Second International Conference on the Software Process*, pages 12–26. IEEE Computer Society, February 1993.
7. Anthony Finkelstein, Jeff Kramer, and Bashar Nuseibeh, editors. *Software Process Modelling and Technology.* Research Studies Press Ltd., John Wiley & Sons Inc., Taunton, England, 1994.
8. Emden Gansner, Eleftherios Koutsofios, Stephen C. North, and Kiem-Phong Vo. A Technique for Drawing Directed Graphs. *IEEE Transactions on Software Engineering*, March 1993.
9. Volker Gruhn. Business Process Modeling in the Workflow-Management Environment Leu. In *Proc. of the Entity-Relationship Conference*, Manchester, UK, December 1994.
10. G. Junkermann, B. Peuschel, W. Schäfer, and S. Wolf. MERLIN: Supporting Cooperation in Software Development Through a Knowledge-Based Environment. In *[7]*, pages 103–129. 1994.
11. R. Kadia. Issues Encoutered in Building a Flexible Software Development Environment. In *Proc of. ACM SIGSOFT '92: Fifth Symposium on Software Development Environments*, pages 169–180, Tyson's Corner, VA, December 1992.
12. T. Katayama. A Hierarchical and Functional Software Process Description and its Enaction. In *Proc. of 11th International Conference on Software Engineering*, pages 343–352. IEEE Computer Society Press, May 1989.
13. Eleftherios Koutsofios and David Dobkin. LEFTY: A Two-view Editor for Technical Pictures. In *Proc. of Graphics Interface '91*, pages 68–76, Calgary, Alberta, 1991.
14. B. Krishnamurthy and N. Barghouti. Provence: A Process Visualization and Enactment Environment. In *Proc. of the Fourth European Conference on Software Engineering*, pages 151–160, Garmisch-Partenkirchen, Germany, September 1993. Springer-Verlag. Published as *Lecture Notes in Computer Science* no. 717.
15. Carlo Montangero and Vincenzo Ambriola. OIKOS: Constructing Process-Centered SDEs. In *[7]*, pages 131–151. 1994.
16. Stephen C. North and Eleftherios Koutsofios. Applications of Graph Visualization. In *Proc. of Graphics Interface '94*, pages 235–245, Banff, Alberta, 1994.
17. Patrick Young and Richard Taylor. Human-Executed Operations in the Teamware Process Programming System. In *Proc. of the 9th International Software Process Workshop*, Airlie, VA, October 1994. IEEE Computer Society Press.

This article was processed using the LaTeX macro package with LLNCS style