Experiments in Process Interface Descriptions, Visualizations and Analyses

David C. Carr[†] Ashok Dandekar[†] Dewayne E. Perry[‡]

[†]AT&T Bell Laboratories
Naperville IL 60566 USA

[‡]AT&T Bell Laboratories
Murray Hill NJ 07974 USA

Abstract. A wide variety of techniques and approaches are needed to understand and improve software development processes. The critical research problem is supporting the move from completely informal process descriptions to a form that includes parts that are at least machine processable. We discuss a series of engineering experiments on process visualization and analysis using simple process interface descriptions. This work grew out of two needs: the need to understand a process's internal structure and the need to understand and improve the process system's architecture. We discuss the various approaches we have taken to understand processes from these two standpoints: we report on the different forms of visualization we have used, both for processes in the small as well as processes in the large, and their resulting benefits; we delineate the various forms of analysis and report how they have played a seminal role in process improvement by providing the quantitative basis of both process understanding and process improvement efforts.

1 Introduction

For the past year and one-half, we have worked to improve our software process descriptions and process structures, as well as the software production processes themselves. We conducted a number of engineering experiments to determine the usefulness of a simple process interface descriptions language as the basis for visualization and analysis. The resulting tool fragments and tools suites have proved to be extremely useful in the understanding and improvement both of our processes and of our process system architecture.

The basis of this effort is a set of on-line manuals for the system development processes and subprocesses. These processes and subprocesses cover the entire range of the software production process from initial marketing interactions with customers through final customer support. They are described informally, but within a highly structured document format. The on-line manuals as a whole occupies about 102 megabytes of disc space and is accessed on the order of four to five hundred times a day.

Our experiments in process description, analysis and visualization began in response to two different needs. The first need was for visualization of the internal structure of a process and its data models to help in re-engineering various development processes; the second need was for an understanding of the effects

on the overall process system architecture resulting from the merge of the development processes of two organizations that produce related products.

The engineering experiments described in the paper test the hypothesis that a very simple process interface description language can provide a significant amount of benefit in understanding processes both by visualization and by analysis. The results of our work have confirmed that hypothesis. Moreover, like a really useful theorem, our approach has provided the basis for a significant number of process improvement efforts.

In the subsequent sections, we describe our experimental approach, present our approach to simple process interface descriptions, discuss and illustrate our various forms of visualization, and delineate the various kinds of analysis that augment the visualization experiments. Finally, we discuss related work and summarize what we have accomplished, what we have learned from these exercises, and where we are going.

2 Experimental Approach

Experimental approaches to building software come in various guises. At one end of the spectrum is the completely rigorous experiment with attention to experimental design, well-defined experimental instruments and metrics, experimental subjects and analysis (as an example see Perry, Staudenmayer and Votta [12]). With this approach, one expects to produce quantified results about the software and its intended use. The primary purpose of scientific experimentation is to determine causality among variables in the experiment. At the other end of the spectrum are engineering experiments. With this approach, one expects informal judgments about the intended use. The primary purpose of engineering experiments is to determine usefulness. That is, we want to establish the appropriateness and utility of the experimental software and the insight it provides into the problems in question.

We take an engineering experimental approach using two complimentary methods. The first is the method on constructing the experimental software. We could call it "rapid prototyping", except that we are not prototyping but producing a complete tool sufficient for the solution needed. However, the emphasis is on rapid construction and evolution. We do this by using shell scripts to glue various existing tools together such as `dot` (a graph drawing tool [7]), sort, join and various tool fragments that operate on various transformations of the process descriptions that have been manipulated by means of pattern-matching scripts in `awk`. The various tools and tool fragments were constructed in anywhere from a few hours to a day or two. This tool-fragment approach enabled us to experiment with a variety of different approaches to both analysis and visualization without a heavy investment in system building.

The second is the method of designing, evolving and evaluating the experimental tools and tool fragments. We did this in a cooperative, incremental and iterative manner. The cooperation was between the users and the builder, proceeding iteratively in much the same manner as in a series of experiments where

hypotheses are tested and evaluated and new hypotheses formed to be tested in new experiments. We proceeded incrementally adding aspects and features to the various tools as we learned what worked and what did not.

The result of this experimental approach has been a suite of related visualization and analysis tools and tool fragments that have been seminal in guiding the understanding of our existing processes and their architectural structure and that have been essential in providing the basis for process and process architecture improvement.

3 Simple Process Interface Descriptions

The basis of our experiments in visualization and analysis has been a simple form of process interface description — that is, a simple approach to describe process interfaces that is accessible to non-programmers as well as to programmers. Some of the people working on process re-engineering effort were not programmers, but were rather process domain experts. A description format that was simple and straightforward was necessary to enable them (the non-technical people) to build and understand their process and data models. Surprisingly, this approach has also been extremely effective for more sophisticated users as well.

From the standpoint of analysis and visualization experimentation, this descriptive approach has proved to be an exceedingly useful basis for quickly building sets of tool fragments. We have built visualization tools for the internal structure of process, including their associated data models, as well as for the architectural structure of the entire set of processes and subprocesses. Furthermore, we have provided analysis suites for task, subprocess, process, and cluster interfaces, various forms of cost and interval analyses, data model redundancy, and others.

Our simple description language (which we call Mini-Interact; see [10] for a description of Interact) consists of records with keywords (either explicit or implied) and one or more fields all separated by tabs. For example, a process interface description consists of a process identifier record followed first by one or more input records and second by one or more output records. The input records consist of the input artifact name and the supplier name; the output records consist of the output artifact name and the customer name. Note that this form of description remains the same whether it is a task, subprocess, process or processcluster. The only difference is the initial keyword.

TAB is used as the field separator: ⇒

Proc: ⇒ *process/subprocess-name*
Input: ⇒ *input-name* ⇒ *supplier-name*
⇒ *input-name* ⇒ *supplier-name*
. . .
Output: ⇒ *output-name* ⇒ *customer-name*
⇒ *output-name* ⇒ *customer-name*

	. . .	
Proc:	*process/subprocess-name*	
Input:	*input-name*	*supplier-name*
	input-name	*supplier-name*
	. . .	
Output:	*output-name*	*customer-name*
	output-name	*customer-name*
	. . .	

Thus a process or subprocess interface might look like the following example of a system architecture subprocess.

Proc:	System Architecture Subprocess	
Input:	Request	FEP Overview Process
	Request	Requirements Process
	Request	Project Management Process
	Architecture Checklist	Architecture Checklist Subprocess
	Architecture Volume	System Architecture Subprocess
Output:	Architecture Volume	Requirements Process
	Architecture Volume	Software Design Process
	Architecture Volume	Hardware Development Process
	Architecture Volume	System Architecture Subprocess
	New/Updated Feature	FEP Overview Process

In transcribing the initial set of process interfaces, we noticed the following things about the artifacts and their suppliers and customers. First the artifacts were of widely varying granularity, ranging from a request (as in the example above) to the entire system. Obviously, this is an important architectural consideration that must be addressed in the ensuing improvement efforts.

Second, there was a wide range of characterizations of these artifacts, ranging from whole artifacts to pieces of artifacts and from plain descriptions to heavily qualified descriptions:

- Whole versus Parts — one might find the *Design Document* referenced as output from one process and the *Low Level Design Chapter* referenced as input by another.
- Generic versus Particular — a *Modification Request* is the generic form while a *Product* Modification Request is a particular kind of MR;
- By Characterization — Software Modification Requests *from previous releases* ; and
- By State — an *Open* Modification Request.

As we will see below in the analysis section, these various ways of referring to essentially the same artifact causes problems. As a result, we extended the

artifact notation to describe a base artifact and to qualify it with one or more characterizations. In this way, we have a formal way of defining the base artifact and of qualifying it in a standard way. For example, the artifacts mentioned above would be described as follows.

Design Document
Design Document: Low Level Design Chapter
Modification Request
Modification Request: Product
Modification Request: Software; Previous Release
Modification Request: Open

Third, the suppliers and customers ranged over a surprisingly wide variety of possibilities. For example, we found the following as suppliers and customers:

- Processes and Subprocesses — these can be checked against the of official list of processes and subprocesses;
- Specific Roles and Classes of Roles — a specific role might be an *end user* where a class of roles might be the *product users* ;
- Organizations/Groups by Name/Descriptions — *Systems Engineering* is a specifically named organization whereas *account teams* are groups characterized by a description; and
- Collections of Roles and Groups — *field course participants* and *AT&T Business Units* are collections of roles and groups, respectively.

It is our observation that, except for a few cases such as customers, only processes and and subprocesses should be the suppliers or customers of process artifacts. We offer the following reasons for this. First, only processes and subprocesses are defined. One can look at the descriptions of the process to find out how the artifact is produced or consumed. With roles and organizations it is not possible to determine this. Second, people in the context of roles and organizations are executing some process. It is by means of these processes that roles and organizations produce and consume artifacts. Finally, the distribution of process artifacts to those executing a particular process is a matter to be decided by the that process definition itself, and not by external processes. It is a matter of encapsulation, of abstraction, of the appropriate separation of allocation and scheduling concerns.

4 Visualization — Comprehension

The initial form of visualization was needed as a part of a process reengineering effort. Our goal in this form of visualization is to aid in the comprehension of a single process — that is, to understand the relationships among the various task steps, among the tasks and their artifacts, and between the artifacts.

Initially, we differentiated the various elements in the process by different shapes and different styles of lines: ellipses represent task steps, diamonds represent external processes, boxes represent artifacts some of which may be decomposed into component parts, solid arrows indicate the input of an artifact to a task step, and dashed arrows represent the output of an artifact from a task step. The example in Figure 1 illustrates this early form of process visualization.

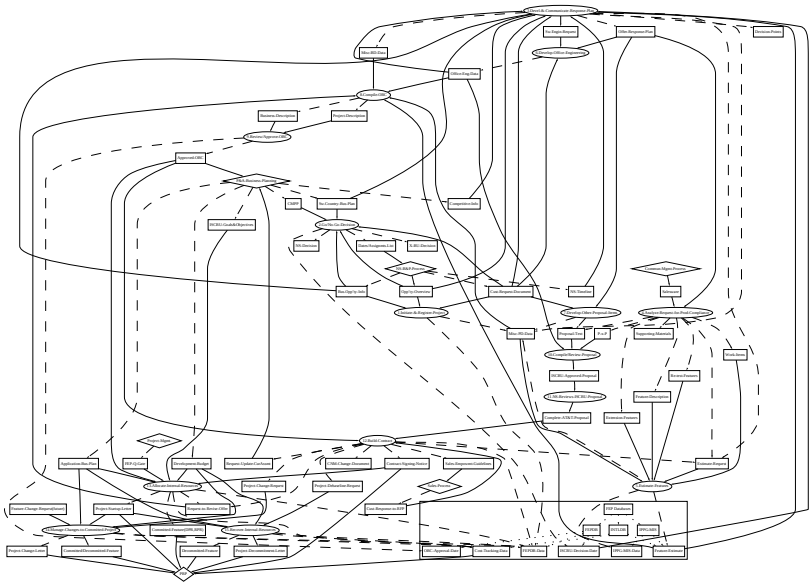

Fig. 1. Figure 1: An Example Process Visualization

The next step towards a more complete comprehension was the separation of the artifact structure from the task model. The desire of the users was to model the data separately from the tasks that manipulated it. We then extended the artifact description language to provide the typical kinds of data relationships: component parts, alternative parts, derived parts, shared components, n-ary relationships between components, etc. Furthermore, the attributes of each data component in the model were shown as subcomponents of the component. Derivation is indicated by a dashed line; sharing is indicated visually by multiple arrows, etc.

Having the capability to define various data models which interact with each other, we then provided the ability to compose data models together for both visualization and analysis purposes. Composing the models provided a visual means of determining what was common among the models. We also provided commonality analysis to provide the same information without drawing the models.

An important part of our support for comprehending processes is our desire

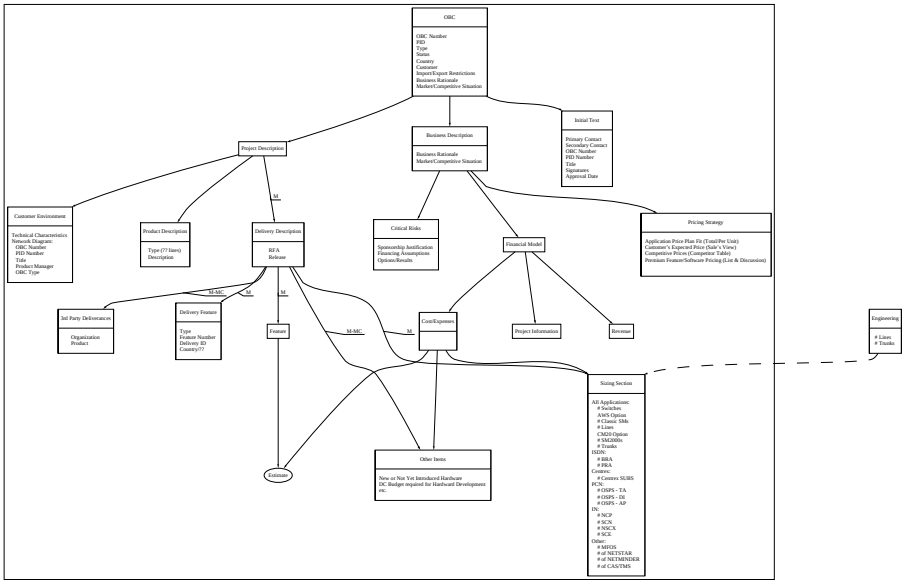

Fig. 2. Figure2: An Example Data Model Visualization

to simplify and improve those processes. An important part of the necessary background for process understanding is the cost associated with various process components, be they processes, task steps or artifacts. We added auxiliary descriptions to define various kinds of cost data that could be associated with those process components. See the analysis section below for a more complete discussion of the cost analysis. In one extension of our task visualization we extended the descriptions to incorporate the cost data into the visualization of the task model.

Further steps in the composition of the data and task models have led to the ability to color various components of the process models. We do this in two steps: first we characterize the components in a separate table. For example, we might characterize task steps as adding value to the customers or to the business, or as adding no value. We then color these different characterizations. This enables us to visually distinguish various aspects of the models.

All in all, the data and process model visualization has been very successful in several different re-engineering and improvement exercises. We continue to experiment with various ways of visually clarifying the relationships among the objects in both data and process models.

5 Visualization — Architectural Complexity

The various experiments at visualizing the internal structure of processes led to the experiments to visualize the architectural structure of the entire set of software development processes. The immediate trigger of this direction came from

merging two complete sets of processes from two organizations that produced similar products. We needed to understand whether resulting merges, done on a process by process basis, produced a coherent whole — that is, did we have a coherent global structure as a result of the local evolutions.

The initial path that we took here was to start with a subset of the merged processes to use as an example basis for visualization and analysis, and then moved to the entire set of processes. We modified our initial task description syntax slightly to refer to processes instead of tasks. Using this amended syntax we generated two large graphs to try to understand the overall process structure. Figure 3 shows the input output relationships between the sources and sinks in the system. With the base set of processes, this graph's primary message was "significant complexity".

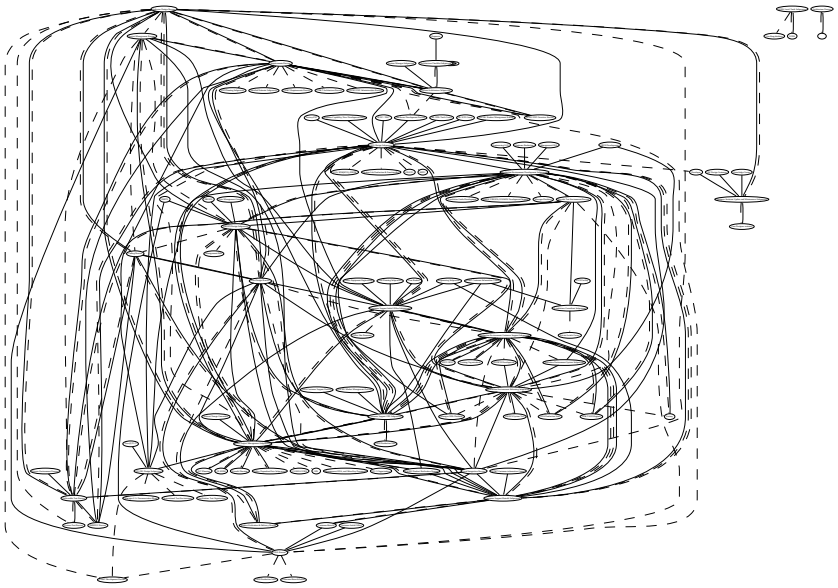

Fig. 3. Figure 3: Input/Output Relations among a Subset of Sources and Sinks

We then generated a visualization of the artifact relationships among only the processes and subprocesses. Figure 4, while still clearly indicative of complexity, was of much more use in gaining an understanding of the overall structure. This view became our canonical "big picture" of the process architecture in which the entire structure is presented at once. With the full set of processes and subprocesses, the big picture requires a graph that is 6 feet high and 25 feet long and even then the print is quite small (this is partly because long process and artifact names, but for the most part because of the sheer volume of processes and artifacts).

The interesting thing about this big picture, which clearly indicates that

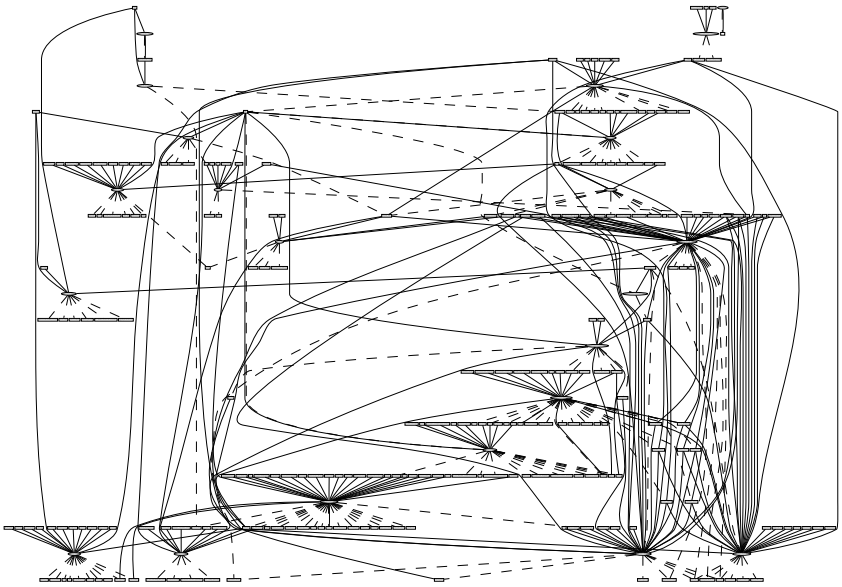

Fig. 4. Figure 4: A Subset "Big Picture" — Processes, Subprocesses and Artifacts

things are far to complex, is that it has been an amazingly effective catalyst for various important process improvement efforts. First, it is effective in showing the importance of a more coherent process architecture. Second it is effective in generating useful observations about the current architectural structure. And third, it has been fundamental in showing that we do not have an effective mechanism for managing process-in-the-large problems.

Having been encouraged by Al Barshefsky to "build it and they will come", we built the big picture and they did in fact come to look at it. The visitors ranged from developers all the way through the top levels of management. Some of the more cogent observations that came from these visits are listed below. Needless to say, these observations have provided some of the foundations for subsequent improvements.

- There is no clear path through the project.
- Project management is the center of the picture.
- A large number of artifacts are consumed internally.
- The core software development processes (requirements through coding and testing) have the simplest interfaces.
- The core software development processes appear to be sequential.
- There is a wide variance in the level of detail used to describe artifacts.
- There are an enormous number of input/output mismatches.
- There is a very high degree of fan in and fan out.
- There are no feedback loops.

In looking at the overall set of processes and how they interact with each

other, we noted the following kinds of architectural relationships that exist in this process system.

- Clearly, there are many processes that are completely *independent* of each other.
- Some processes are dependent on other processes for input — that is, they are *artifact dependent.*
- Some processes cooperate with others and mutually iterate (as for example do design and coding processes) — they are *cooperating and concurrent.*
- Some processes, such as the modification request process, are *subroutine like* — that is, the current process is suspended and the MR process executed.
- Some processes, such as project management, function more as a *monitoring process* than as an dependent, cooperating or subroutine-like process.

See [4] and [11] for more complete discussions about process architectural considerations.

As with most well-defined quality programs, we have quality gates that mark important states in the development process. These tend to give the appearance of a waterfall process even thought everyone recognizes that there is far more going on both iteratively and concurrently. These gates, however, do represent some agreed upon state of either the process or the product and are important in determining progress in a project.

To try to get a better feel for what kinds of things must be going on concurrently, we provided a process dependency tree. While this is a static representation of the processes, it provides an important reminder for the large amount of work that must precede any particular quality gate in the process. Figure 5 shows the processes that must have taken place for a sample process to be able to begin: the sample process requires input from 5 processes which in turn require input, etc. We have terminated circularities in order to create a finite tree. An interesting observation derived from creating several samples was that their general appearance was not all that much different. The lesson here is that production processes, especially those considered to be the core processes, are significantly dependent on a large number of supporting processes.

6 Analysis

The visualizations gave us abundant evidence that the process interfaces were not consistent with each other. While each process interface description has both input/supplier and output/costumer sections, it was clear that these descriptions did not match in many cases.

Using the same descriptions that we used to generate the process visualizations, we incrementally evolved a set of analysis and summary tool-fragment suites to perform process interface analyses and summarizations.

- Input/Output consistency/mismatch analysis
- Process and artifact summary analysis

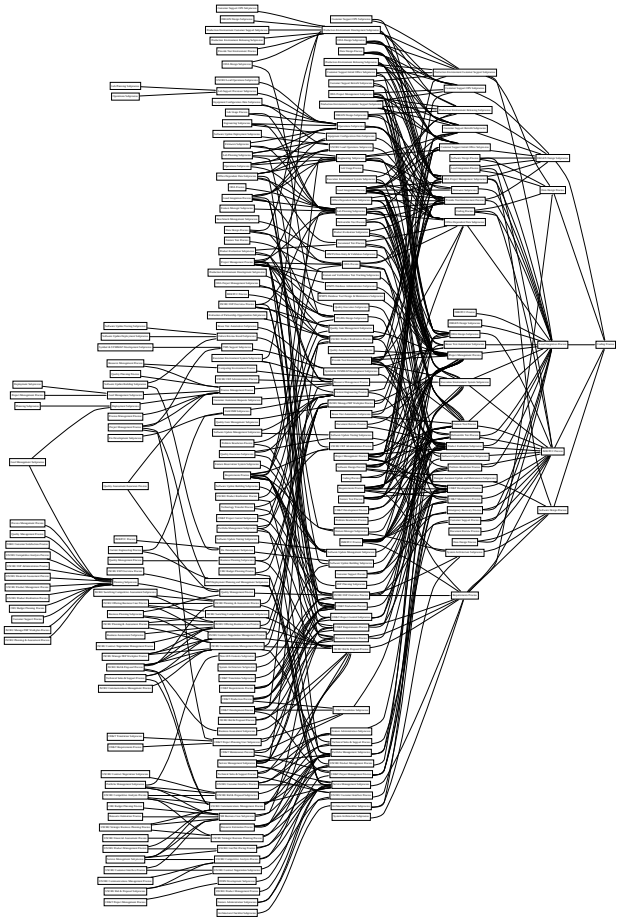

Fig. 5. Figure 5: A Sample Process Dependency Tree

- Artifact definition/usage summary
- Summary of undefined processes/subprocesses referenced
- Alphabetized artifact and source/sink lists
- Artifact and Source/Sink reference count lists and summaries

While it might be noted that we only needed the input and output artifacts to generate the pictures that we have used so far, it is in the analysis that the utility of the redundant supplier and customer information becomes obvious. Since connecting elements [13] in process architectures may be entirely informal (as opposed to the formal and automated connecting elements in product architecture), processes must be much more proactive in distributing artifacts. For this reason, the redundant information is particularly important.

The primary analysis on process interfaces checks whether the input and output specifications match. This is exactly analogous to signature checking in

programming and programming-in-the-large languages. In our case, we have no formal name definition facilities, but we do have the name qualification facilities described above. Process interface checking then is basically name (string) matching. As simple as this approach is, it is sufficient for the problems at hand. Once we get to the point where we have a consistent and common vocabulary, we will then need more interface information than we currently have. Obviously, we then take the next step towards complete formalization.

We delineate the complete set of possible errors in the supply and use of process artifacts and whether those artifacts are supplied or used by defined or undefined processes. The first two errors are those found when analyzing a process's input and the last two are those found when analyzing a process's output.

- The input and output specifications match.
- Output was expected from a particular process as input but not provided by that process.
- Output was provided from a process but was not expected as input from that process.
- Output was provided to a process but the output was not expected as input by that process.
- Output was expected by a particular process as input but not provided to process.

For example, We might have a process definition with the following analysis results.

System Architecture Subprocess
Input:
Architecture Checklist Chapter
matches the output from
Architecture Checklist Subprocess
Architecture Investigation Request
provided from but not expected from
Requirements Process [defined]
Request
expected from but not provided by
IS FEP Overview Process [undefined]
expected from but not provided by
Requirements Process [defined]
Output:
Architecture Volume
provided to but not expected by
Overall Hardware Development Process [undefined]
matches the input to
Software Design Process
New/Updated Feature
expected by but not provided to

These same inputs and outputs are summarized in the definition and usage summary in the following way.

Architecture Checklist Chapter
- Defined in Processes:
  - Architecture Checklist Subprocess
- Used in Processes:
  - Architecture Checklist Subprocess
  - System Architecture Subprocess

Architecture Investigation Request
- Defined in Processes:
  - Requirements Process
- Unused

Architecture Volume
- Defined in Processes:
  - System Architecture Subprocess
- Used in Processes:
  - Requirements Process
  - Software Design Process
  - System Architecture Subprocess

Request
- Undefined
- Used in Processes:
  - Architecture Checklist Subprocess
  - System Architecture Subprocess

We produce the summary of undefined processes and subprocesses referenced by comparing all customers and suppliers that have the term process or subprocess in them with the official list of processes and subprocesses. The alphabetized list of both artifacts and customers and suppliers is useful for finding those cases where mismatches are due to simple spelling errors. The reference count lists and summaries are used in various ways. The most obvious is to find those elements that are referenced only once — that is, that are either undefined or unused. Another is to find those customers and suppliers that are the most referenced. Given the observation from looking at the big picture that project management is the center of the process architecture, it is not surprising to note that project management is the most referenced customer and supplier by a factor of two over the next most referenced.

The process and artifact analysis summary provides summary data on the number of processes, inputs and outputs, the breakdown on the customers and suppliers, the number of artifacts, how many are defined and used, and the summary of the input/output analysis. We note in passing that the minimum

numbers of inputs and outputs is not particularly meaningful here as they reflect some processes that were named but for which no inputs or outputs were defined. The sample below reports the summary of an arbitrary subset of the system processes.

Total number of inputs:	301
Total number of outputs:	273
Average number of inputs:	12.04
Average number of outputs:	10.92
Minimum number of inputs:	2
Maximum number of inputs:	35
Minimum number of outputs:	1
Maximum number of outputs:	35
Total distinct sources/sinks:	206
Total distinct process references as source/sink:	99
Total distinct subprocess references as source/sink:	10
Total distinct other references as source/sink:	97
Total Artifacts:	423
Total Undefined:	229
Total Defined:	194
Total Defined but Unused:	178
Total Defined and Used:	16
Total number of matches:	7
Total number of expected-from:	300
Total number of provided-from:	26
Total number of expected-by:	272
Total number of provided-to:	14

Two interesting facts are represented in this summary: first, the large number of non-process or subprocess references as suppliers and customers; second, the exceedingly large number of undefined and unused artifacts. We did a manual analysis of the sources and sinks with the following results. Of the references to processes and subprocesses, approximately 40% were correctly named, 20% were slight variants, 20% were completely misnamed, and 20% were unknown or exhibited some other problem. Of the remaining sources and sinks, about one third were possible references to processes (though they did not contain the term "process" or "subprocess"), 45% were references to roles, 35% were references to organizations.

With artifacts, since we do not have a canonical list of agreed upon artifacts as we do for processes and subprocesses, we can say only that about 10% of the

non-matches are due to variants in spelling. This still leaves a large number of undefined and unused artifacts unaccounted for. We conjecture that these are the results of local definitions and a lack of global agreement on the necessary artifacts for the process system.

In summary, we have a set of analyses and summaries that can be applied to the entire set of processes, to a subset of them, or to a single process viewed in the context of the entire set. These analyses have provided us with a quantitative base of data upon which to build a successful improvement program as well as the necessary tools to aid in the improvement processes itself.

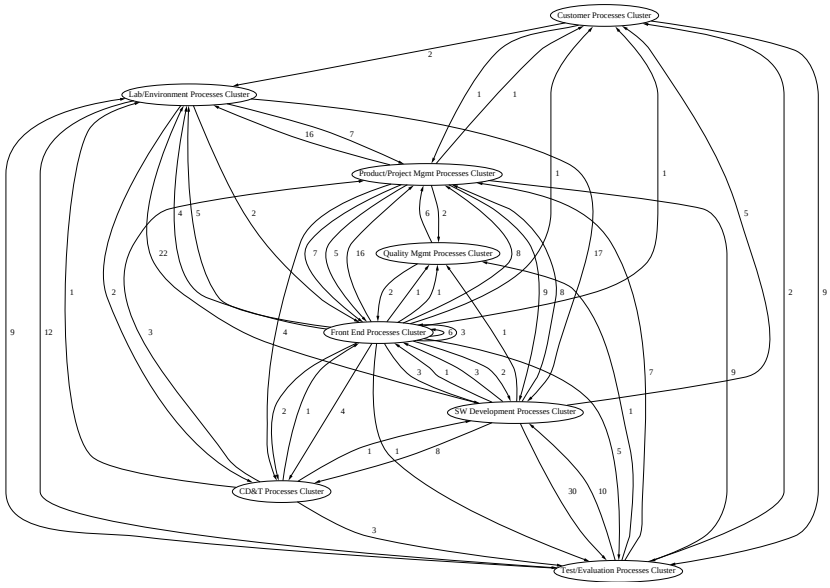

Fig. 6. Figure 6: Process Clusters with I/O Counts

7 Visualization — Architectural Abstraction

One of the main results of both the big picture visualization and the process-in-the-large analysis has been an effort to bring coherence to this process architecture. We are in the process of doing this by partitioning the processes and subprocesses into domain related clusters. The purposes of this clustering are threefold: first to partition the sheer volume of process descriptions into manageable chunks; second to create a useful architectural abstraction of the entire set of processes; and third to divide the improvement effort into that within clusters and that among clusters.

On the basis of a process to cluster mapping table and the process interface descriptions in Mini-Interact, we automatically generate the process cluster

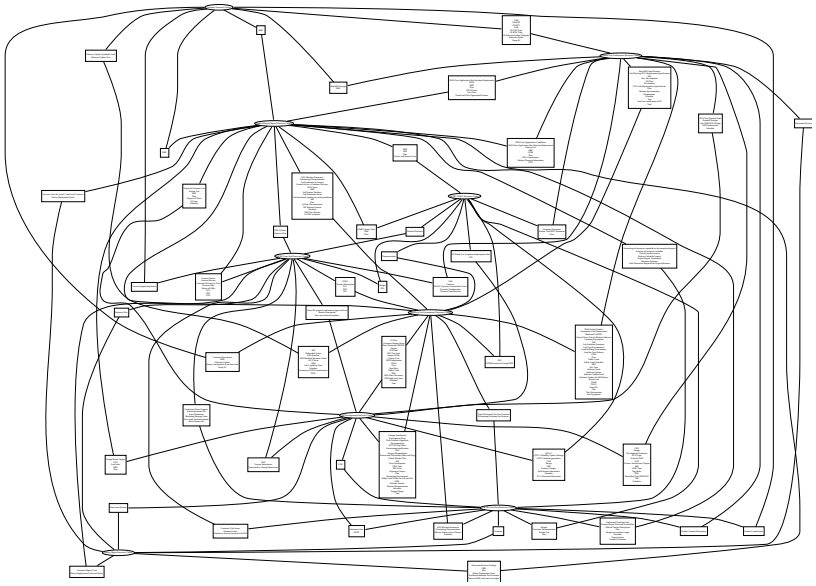

Fig. 7. Figure 7: Process Clusters with Artifact Aggregation

interfaces. These cluster interfaces are identical in form to their subsumed components and thus subjectable to the same visualizations and analyses that we have described for processes.

We have, however, extended our visualizations to provide more of an architectural abstraction approach. The first cluster visualization (Figure 6) was similar to the input/output relations that we used for the subset (Figure 3) with the extension of labeling the input and output arcs with the number of artifacts that are input or output between those two process clusters. Given that we have now on the order of 10 clusters, this makes a very understandable architectural abstraction.

The second cluster visualization expands the initial cluster visualization by providing a list of the artifacts as part of the arc between clusters. Thus we aggregate the artifacts that are passed between clusters (Figure 7) and get a more comprehensible architectural view of the clusters and their input/output artifacts.

We note that in generating these cluster visualizations we have ignored all the sources and sinks that are not defined processes in order to remove some of the clutter from the picture. This decision is defensible on the basis that, as we mentioned earlier, almost all of the input and output ought to be between processes, or in this case, between clusters.

8 Related Work

There are several different strands of related work. One strand is that of workflow tools (of which AT&T GIS' ProcessIT [14] is a representative example). These tools generally take an interactive graphical approach and are very useful in modeling workflow as static tasks and products. They tend to have hierarchical or decomposition facilities as well. Thus at one level, there is a fair amount of similarity to the approach that we have reported here. However, there are substantial differences. First, our approach is based on an underlying textual process description from the visualizations, analyses, and clusterings are generated. This provides us an inherently more flexible basis for our experimentation. Second, we have yet to see whether these tools, as with most of the current process support tools, scale to the magnitude that we have represented here. We know of no process related tools that are set up to be used by multiple people working in a distributed manner to build a large set of interacting process or workflow descriptions.

Another strand is that of process-in-the-large facilities in various process formalisms, such as those in SLANG [3], EPOS [8], ALF [9] and APPL/A [15]. These approaches provide various ways to represent activities and use activities to build hierarchical structures. Interact [10] also has a notion of activity. However, process activities are related to each other in a non-classical way: they are related to each other by means of the policies that are used in the preconditions and the results — that is, there is an implied partial ordering of these activities. Interact further has a notion of a parameterized model in which models and activities can also appear as arguments. None of these formalisms has been used on large process systems. Thus, while there are good arguments for their utility as applied to large interacting processes, none have as yet been tested. Moreover, none give us the flexibility to address the various architectural relationships among processes that we delineated above.

The last strand is that of applying process support mechanisms to very large process systems. The only other reported account of such an effort is that of the PROMESSE project [1] using PROCESS WEAVER [6]. Their model consisted of three components: a product model, a description of process roles, and a set of lifecycle phases and subprocesses. Our work reported here supports the first and last of these aspects: both explicit and implicit product models, and of course, processes and subprocesses. There is a similarity in intent: create a specification of existing processes in order to understand the process system. The difference between the two lies in what is done with that specification. In the PROMESSE project, they implemented the specification in PROCESS WEAVER and were then able to validate it with various scenarios. In our project, we used the specifications as basis for analysis and visualization. One of our plans is to expand our cost data together with the specifications to build a queueing model with which to do various kinds of simulations (in effect, validation).

The first important difference in our approach from all of the above is the ability to generate various levels of interface descriptions, ultimately on the basis of the bottom level task descriptions. From those task interfaces, we generate

process and subprocess interfaces, and from those interfaces we generate cluster interfaces. The second important difference is the interface analysis. We believe that we are the first to apply interface analysis to processes-in-the-large.

9 Summary and Future Directions

Our work has resulted in a number of different research contributions. First, we have provided an easy to understand and use process interface description language which we call Mini-Interact. With these descriptions and existing tools as the basis, we have established an effective tool creation and evolution paradigm.

Second, this paradigm has been the basis for our experiments in process visualization, analysis and interface generation. While we would not claim that our visualizations are unique (similar visualizations may be found in software understanding, for example), we do claim that we have found several that provide various kinds of insights needed to understand both interface relationships and architectural structure. Moreover, as we mentioned, we believe that we are the first to provide process-in-the-large interface analysis.

Third, we have provided insight into various process interface problems such as the various kinds of customers and suppliers referenced in the interfaces, the structure of process artifacts, and the relationships among processes.

Fourth, through our experimentation, we have demonstrated the effectiveness in our approach to supporting process understanding and comprehension. In addition to their immediate utility, they are being used as the basis for a variety of important process improvement efforts.

Three of the most important of the various improvement efforts that have arisen from this work are the following. First, we have successfully completed a process architecture study looking at the sources of our architectural problems, their underlying root causes, and various countermeasures [4]. Second, we are in the midst of several process simplification experiments in which we use various techniques to simplify and improve a complex subset of our processes [5]. Finally, we have launched an overall process system improvement effort based on the process clusters.

Our future plans include 1) incorporating the visualization and analysis as a standard part of the management of process improvement and evolution; 2) providing a graphical and interactive front end (in Xtent [2]) that will enable one to explore the architectural structure and correlate various analyses with that visual structure; and 3) include various interval, cost and quality measures to aid in deriving a queueing model of the system or components of the system for various kinds of analysis and simulations.

Acknowledgements

Mary Ellen Biell and Cindy Stach were instrumental in the early experiments with process and data modeling. Al Barshefsky has been unflagging in his support of the architectural visualization and analysis. Mary Zajac provided many

of the insightful observations about the big picture. Larry Votta as a colleague in other process related experiments has provided insights for this work as well.

References

1. J-M. Aumaitre, M. Dowson, and T-R. Harjani: Lessons Learned from Formalizing and Implementing a Large Process Model. Third European Workshop on Software Process Technology, Villard de Lans, France, February 1994, Springer-Verlag.
2. Doug Blewett, Scott Anderson, Meg Kilduff, and Mike Wish: X Widget Based Software Tools for UNIX. USENIX Winter 1992, January 20-24, 1992, pages 111-123.
3. S. Bandinelli, A. Fugetta and S. Grigolli: Process Modeling In-the-Large with SLANG. Second International Conference on the Software Process, Berlin, Germany, February 1993, IEEE Computer Society Press.
4. Ashok Dandekar, Dewayne E. Perry: Experience Report: Barriers to an Effective Process Architecture - Extended Abstract. AT&T Software Symposium, Holmdel NJ, October 1994.
5. Ashok Dandekar and Dewayne E. Perry, and Lawrence G. Votta: Experiments in Process Simplification - Extended Abstract. AT&T Software Symposium, Holmdel NJ, October 1994.
6. Christer Fernstroem: PROCESS WEAVER: Adding Process Support to UNIX. Second International Conference on the Software Process, Berlin, Germany, February 1993, IEEE Computer Society Press.
7. E.R. Gansner and E. Koutsofios and S.C. North and K.P. Vo: A Technique for Drawing Directed Graphs. IEEE-TSE 19:3 (Match 1993).
8. C. Lui and R. Conradi: Process Modeling Paradigms: an evaluation. First European Workshop on Software Process Technology, Milan, Italy, May 1991, Springer-Verlag.
9. F. Oquendo, J. Zucker and P. Griffiths: The MASP Approach to Software Process Description Instantiation and Enaction. First European Workshop on Software Process Technology, Milan, Italy, May 1991, Springer-Verlag.
10. Dewayne E. Perry: Policy-Directed Coordination and Cooperation. Seventh International Software Process Workshop, Yountville CA, October 1991.
11. Dewayne E. Perry: Issues in Process Architecture. Ninth International Software Process Workshop, Airlie VA, October 1994
12. Dewayne E. Perry, Nancy A. Staudenmayer and Lawrence G. Votta: Understanding Software Development Processes, Organizations and Technology. IEEE Software, July 1994.
13. Dewayne E. Perry and Alexander L. Wolf: Foundations for the Study of Software Architecture. ACM SIGSOFT Software Engineering Notes, 17:4 (October 1992).
14. Process IT Product Brochure. AT&T GIS.
15. S. Sutton, D. Heimbigner and L. Osterweil: Language Constructs for Managing Change in Process-Centered Environments. Fourth ACM SIGSOFT Symposium on Software Development Environments, Irvine CA, December 1990. ACM SIGSOFT Software Engineering Notes, December 1990.

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