Analyzing Processes for E-Government Application Development: The Emergence of Process Definition Languages

Leon J. Osterweil
Charles M. Schweik
Norman K. Sondheimer
Craig W. Thomas

Leon J. Osterweil is Professor in the Department of Computer Science, and Dean of the College of Natural Sciences and Mathematics at the University of Massachusetts, Amherst. He has served as Associate Editor of ACM Transactions on Software Engineering, is a Fellow of the Association of Computing Machinery (ACM), and is recipient of the Outstanding Research Award from the ACM's Special Interest Group on Software Engineering (E-mail: ljo@cs.umass.edu).

Charles M. Schweik is Assistant Professor in the Department of Natural Resources Conservation and the Center for Public Policy and Administration at the University of Massachusetts, Amherst. His research focuses on public sector information technology, environmental management and policy, and the intersection of these domains.

Norman K. Sondheimer is the Dean's Executive Professor in the School of Management at the University of Massachusetts, Amherst. He is co-Director with Professor Osterweil of the University's Electronic Enterprise Institute (EEI) and investigates fundamental issues that underlie digital government, electronic commerce and the virtual enterprise (E-mail: sondheimer@som.umass.edu).

Craig W. Thomas is Associate Professor in the Department of Political Science and the Center for Public Policy and Administration at the University of Massachusetts, Amherst. He is the author of *Bureaucratic Landscapes: Interagency Cooperation and the Preservation of Biodiversity* (MIT 2003), and articles on collaborative governance, environmental management, and public trust in government agencies (E-mail: cthomas@polsci.umass.edu).

Address correspondence to: Charles M. Schweik, Department of Natural Resources Conservation and Center for Public Policy and Administration, 217 Holdsworth Hall, University of Massachusetts, Amherst, Amherst, MA 01003 (E-mail: cschweik@pubpol.umass.edu).

Journal of E-Government, Vol. 1(4) 2004
Available online at http://www.haworthpress.com/web/JEG

Digital Object Identifier: 10.1300/J399v01n04_05

ABSTRACT. E-government developers need to clearly understand processes they are automating and ensure that automated processes are defect-free. This paper introduces readers to Process Definition Language (PDL) technology that provides rigor and precision over traditional forms of process documentation. We report our experience documenting license renewal processes for application in the Mass.gov portal. The PDL helped analysts identify inconsistencies and errors in natural language-based documents that were guiding system development. This case provides an initial demonstration of the benefits PDLs can bring to e-government application development. We conclude with a discussion of the current limitations of PDLs and a discussion of computer-based analysis approaches that will likely emerge in the future.

KEYWORDS. Process definition language, systems analysis, e-government application development, trusted processes, process documentation, workflow

INTRODUCTION

Readers of this journal are well aware of the rapid deployment of e-government worldwide (Moon, 2002; Ho, 2002, West, 2002a; West, 2002b; Seville European Council, 2002). The services being automated, such as license registration, electronic payment, filing of documents, and information provision, often involve intricate collaborations between computer systems, government staff, and the citizenry who are the users and recipients of these services. Government managers and e-government application developers are faced with the challenge of demonstrating that these applications are error-free.

This is not all that different from traditional information systems development in organizations, except that in the context of e-government applications are widely seen and used by people outside the agency. And there is one other important distinction: in most e-government contexts, the manual process is not replaced, but is duplicated online. In these settings, e-government application developers, analysts and government managers need tools that can help them analyze existing or pro-

posed online (and offline) processes to ensure they are well understood and function as desired.

The main objective of this paper is to introduce readers to the emerging field of Process Definition Languages (PDLs). The paper emphasizes the utility of these tools for public administration in general, and e-government application development in particular. PDLs are formal languages that provide analysts with the ability to articulate organizational processes in a rigorous, precise, complete, and clear manner. PDLs are most distinguished by the rigor with which they themselves are defined, and the completeness of the definitions of the processes they are used to define. While particular process definitions vary, in general most PDLs provide a means to specify who (a person or position) or what (such as a computer program) is responsible for conducting a particular task; when the task is to be undertaken; when parallelism is possible; what kinds of resources are required (e.g., people, computer programs); where data used in the process are stored; what kinds of activities or sub-steps of the process occur; what other people or processes need to know about the outcomes or intermediary stages of the process; and what kinds of outputs are produced. While PDL technology are still emerging, examples of existing PDLs include: "Adele" (Estublier, 1985; Belkhatir, Estublier and Walcelio, 1991; Belkhatir, Estublier and Walcelio, 1993), "Slang" (Bandinelli, Fuggetta and Grigolli, 1993; Bandinelli and Fuggetta, 1993) and "Marvel" (Kaiser, Barghouti and Sokolsky, 1990; Heineman et al., 1992).

Because of the precise nature of PDLs, errors or inconsistencies in process logic can be identified more readily than if the process were articulated using other methods of process documentation that are popular today, such as data flow diagrams, use cases, or natural language (e.g., English) descriptions. The precision of PDLs brings additional benefits, such as comparability and reuse. Through explicit representation, coupled with visualization or graphical features, PDLs provide the analyst with tools to enhance comparison of processes. In the e-government context, this can assist in ensuring that online and offline processes that are intended to be identical are indeed so. This documentation adds another benefit: all or parts of a trustworthy or a proven-to-be-efficient process articulated in a PDL can be readily copied and reused in other areas.

With the goal being to introduce readers to PDLs and their utility in e-government application development, the paper begins by providing a definition of a PDL and a more complete discussion about the main benefit of PDLs: *enhanced process analyzability*. Next, we present an em-

pirical example of a PDL in action; we describe the PDL we used (called "Little-JIL," created by computer scientists at the University of Massachusetts); and our experiences using this PDL to validate the designs of online license renewal processes being implemented in the Commonwealth of Massachusetts' e-government efforts. To demonstrate the main benefit of analyzability, we provide some comparisons between more traditional natural language use cases and our experience with Little-JIL formalisms for license renewal processes. But the analyzability features we report here only begin to demonstrate the potential PDLs can bring to public administration in the analysis of online (and offline) processes. We close the paper with a discussion of where we expect PDLs and PDL-based process analyzability to go in the future, based on what already is occurring in the field of software engineering.

PROCESS DEFINITION LANGUAGES, E-GOVERNMENT APPLICATION DEVELOPMENT AND ANALYZABILITY

The issue of understanding and documenting organizational processes is not new. Indeed, the eminent sociologist Max Weber, writing before the emergence of computers, identified written documentation as one of the core characteristics of bureaucracy (Weber, 1946). For Weber, written documentation was important because it separated the working life from the private life of public officials. During the 20th century, the documentation of processes became a routine part of organizational life for other reasons, as well. For example, some organizations operating hazardous systems, such as nuclear power plants, extensively documented their operating procedures as a means for searching out the root causes of potentially catastrophic errors to prevent such errors from occurring (La Porte and Thomas, 1995).

Documenting public-sector processes is difficult. Even the most simple-appearing processes are remarkably intricate. The intricacy comes in part from the need for a process definition to articulate not just the activities that are to be performed, but also the rights and responsibilities of the agents (human or computer) that must carry them out, the resources these agents are to be granted, the time limits on process execution, and the appropriate responses to deviant or alternative conditions that might arise. Even prior to the emergence of e-government there was a critical need for public administrators to articulate processes clearly and analyze them for defects, particularly in the case of organizations

operating or overseeing critical systems such as electric power grids. The emergence of Web-based e-government services and the continued need for manual processes to serve constituents who are not yet online makes it even more crucial that public administrators are able to articulate completely and precisely the broad range of process activities, agents, resources and timing issues.

Yet in most organizations processes tend to be described using informal natural language text, with the only parts of processes defined precisely being those performed by computers. In the latter cases, the process definition is captured as executable computer code, which undergoes varying degrees of analysis and testing to ascertain its correctness and performance characteristics. But humans perform large and important aspects of most public sector processes, and these parts are often defined far less precisely and are far less amenable to definitive analysis.

This quest for increasingly effective process definitions has been materially aided by the growth of computer science, which necessitates very precise and complete languages for communication of computational needs to computing devices. Languages for communicating these needs directly to computers ("programming languages") are most familiar. But, in addition, there has been increasing acknowledgement of the need for languages for communicating to organizations and stakeholders the myriad processes needed to conceptualize, design, implement, test, and evolve software systems. Thus, over time, computer scientists and others have created methodologies and languages for specifying these processes and defining software requirements. These methodologies can be generally categorized as (1) dataflow languages, (2) workflow languages, and (3) PDLs. There are no hard and fast definitions of these languages, or rules for categorizing them, and we will not attempt to do so. Rather, we suggest these qualitative characterizations.

Dataflow languages can be placed at one end of a comparative spectrum. They tend to be pictorial, and highly appealing to human intuition and understanding. On the other hand, they tend to be relatively limited in semantic scope (i.e., the range of different aspects covered and descriptive power), and lack rigor in their definitions. That is, it is relatively easy for the analyst to leave something out of the process definition, or to develop a process definition that is too vague or is inconsistent. PDLs fall at the opposite end of this spectrum. They are broadly comprehensive in semantic scope, and follow rigorous rules about what needs to be included in their definitions (Cass et al., 2000).

Because of their rigor, they are amenable to computer-based analysis, although often at the expense of easy comprehensibility by humans. Workflow languages span a spectrum of approaches between these two extremes, some emphasizing rigor and semantics, and others hewing more to the side of human comprehensibility. For example, the "use case" approach to process description (Cockburn, 2001; Schneider and Winters, 2001; Dennis, Wixom and Tegarden, 2002) could be classified in the workflow language category. These are semi-structured descriptions of processes that use a mixture of diagrams, tables and natural language text that are organized in a standard format. (We should note, however, that by creating the PDL distinction, we do not mean to exclude workflow languages–or any languages–that also provide analytic capabilities.)

Improved and formal analyzability of a process is the main advantage a PDL provides over these other process analysis techniques. There are two reasons why PDLs improve the analyzability of a process. First, PDLs are broadly applicable and expressive but at the same time require components of a process to be rigorously and precisely defined. Natural language descriptions do not require this kind of rigor, and consequently can result in descriptions with vague, incomplete, imprecise, or ambiguous language that will eventually lead to errors or omissions in process descriptions.

Second, once analysts overcome an initial learning curve, PDLs provide clear, readable, and understandable process descriptions. There are many ways clarity can be created in a PDL. Graphical depiction of the process is a useful approach, for it is commonly understood that graphics enhance the communication of complex information (Tufte, 2001). Such use of visualization could provide government process inspectors with substantially higher confidence in processes, simply because the inspector can absorb more of the complexity through a visual graphic of the process.

On the other hand, visual inspection of graphical representation of processes is typically not sufficient. Process definitions can require much detail and the graphics can become quite complex. Thus, a visual inspection can improve intuition, but is not sufficient by itself to address the need to deal with volumes of detail. Thus, a second approach to clarity is needed, which can be provided through abstraction–the ability to view the details of a process at incrementally aggregated or summary levels to make things simpler to understand, and to decompose aggregations when needed. Considerable literature exists that describes the pivotal role of abstraction in effective comprehensibility of software (e.g.,

Hoffman and Weiss, 2001; Szyperski, 2002), which is readily transferred to the study of organizational processes.

Currently, natural language descriptions are widely used in the generation of requirements for public sector information systems and the development of e-government. For example, natural language-based use case descriptions were a major part of the requirements definitions for transaction processing being developed for access through the Commonwealth of Massachusetts's Mass.gov portal.

Our main hypothesis in this paper is the following:

> *The rigor and precision of a PDL helps ensure that processes are defined more thoroughly and precisely, facilitating the identification of errors and inconsistencies in natural language-based process definitions.*

Moreover, an additional benefit of processes defined using a PDL over those defined in natural language is that their clarity and abstraction may help to identify process or sub-process comparability and reuse opportunities. In the context of e-government, the identification of reuse opportunities is particularly important for cost savings (e.g., if a computer transaction processing program is written to serve multiple purposes), and for improving the quality and security of online transactions. If reuse opportunities can be identified, it may be easy to apply an already proven secure and functionally correct program to another yet-to-be-developed e-government system.

The next section provides readers with more detail of one such PDL that exhibits these analytic qualities. This is followed by a description of our efforts working with the developers of the Mass.gov portal, where we used this PDL to analyze and inspect some license renewal processes already defined using another process definition approach (i.e., use case descriptions) for defects and reuse possibilities. We present the results of this effort, which support the hypothesis and statements made above.

A PDL EXAMPLE: DESIGNING E-GOVERNMENT LICENSE RENEWAL TRANSACTION PROCESSING

To give readers a better understanding of a PDL "in action" and to demonstrate some of the analyzability benefits we described above, this

section describes our recent experience applying a PDL called "Little-JIL" to support an e-government application development project. Little-JIL is a PDL developed by Alexander Wise, Barbara Lerner, Stanley M. Sutton Jr. and Leon J. Osterweil at the Laboratory for Advanced Software Engineering Research at the University of Massachusetts, Amherst. Little-JIL was applied in 2002 as part of an effort by the Commonwealth of Massachusetts Office of Consumer Affairs to create new online license renewal services through its Mass.gov portal. In this section we (1) provide a brief overview of the Little-JIL language; (2) describe the project, and present one Little-JIL license renewal diagram developed for the project; and (3) present some examples of how this PDL helped human inspectors detect some defects in natural language use case descriptions. At the outset, we should emphasize that Little-JIL is just one PDL to which we had access–there are others, which the interested reader can explore through the references in this paper.

Overview of the Little-JIL Language

The argument we have made is that PDLs improve analyzability because their formal semantics provide precision and rigor. PDLs that also employ graphics and abstraction thereby also improve clarity and comprehensibility. The Little-JIL language provides an example of a PDL with such capabilities. A Little-JIL process is represented as a hierarchical decomposition of process steps and a graphical syntax. Figure 1 shows the various "badges" (or icons) that make up a step, and a step's possible connections to other steps. The "interface badge" at the top is a circle by which this step is connected to a parent step. The interface includes declarations of the "agent" (usually a human or computer) that is to carry out the step, resource requirements of the step, exceptions that may take place during step execution, and messages that may be sent from this step to another step.

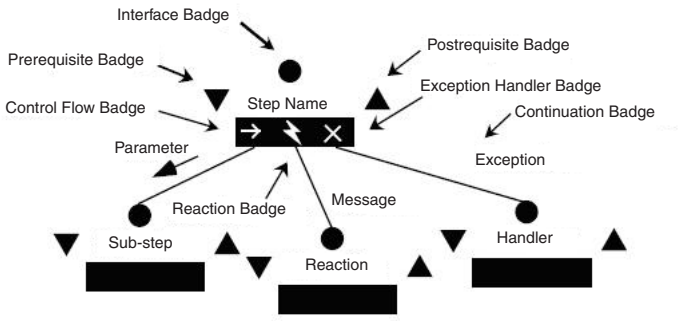
Below the interface badge is the step name. To the left is a triangle called the "prerequisite badge," which is a step that must be successfully completed before this step begins. If the prerequisite is not completed successfully, the step is not allowed to execute. On the right side of the step box is another similarly filled triangle called the "postrequisite badge," representing a step that begins execution immediately after the step completes execution and must also successfully complete for the parent step to be notified of the child step's completion.

Within the box below the step name in Figure 1 are three more badges. From left to right, they are the "control flow badge" (an arrow), "reaction badge" (a lightning bolt) and "exception handler badge" (an X). Child steps (or sub-steps) are connected to the control flow badge. The line connecting a step to a sub-step is annotated with a list of components or artifacts (i.e., data items or collections) needed by the step, and a list of artifacts the step produces. The control flow badge indicates the order in which sub-steps are executed. Little-JIL provides four different control flow badges.

A "sequential control flow badge" indicates that sub-steps are executed in order from left to right, beginning the next sub-step only after the preceding sub-step completes successfully. A "parallel control flow badge" denotes a step that allows its sub-steps to be executed at the same time. Both sequential and parallel steps require all of their sub-steps to be performed. A "choice control flow badge" denotes a step that allows the agent performing the step to choose which single sub-step to execute. A "try control flow badge" denotes a step that identifies alternative ways of performing the step but mandates the order in which the alternatives should be tried from left to right. A choice or try step requires exactly one of its sub-steps to be performed successfully. A step with no control flow badge is executed completely by the agent (human or computer program) to which responsibility for the step has been assigned, with no specified process guidance.

If any sub-step fails to complete successfully, the parent step also fails to complete. This creates a condition that is handled by the actions associated with the "exception handler badge." The "X" in the step box

FIGURE 1. Little-JIL Syntax



of Figure 1 is such a badge, to which one or many exception handlers can be attached. Exception handling may entail execution of a separate step to be performed when the exception is encountered, but must always specify a continuation badge to indicate what to do after the exception has been handled. Little-JIL provides four alternative continuation badges, details of which can be found in Wise (1998). A full process is represented in Little-JIL as a hierarchical set of step boxes. More complicated processes have sub-steps fully described in separate, but connected, Little-JIL diagrams (an example of abstraction). It is important to note that, although these Little-JIL language features have been described (incompletely and tersely) in English, they are also defined completely, precisely, and rigorously by means of a mathematical formalism: Finite State Automata (FSAs) (Aho, Hopcroft and Ullman, 1983). Thus, this language has the rigor needed to qualify it as a PDL.

The Mass.gov License Renewal Project

The Commonwealth of Massachusetts Office of Consumer Affairs has begun offering professional license renewal services over the Web. As part of that effort, they engaged consultants to capture the processes involved using the natural language use case definition approach.[1] This approach differs significantly from the PDL approach in a few ways. Most substantially, use case definitions describe the different views of a process from the perspectives of its various users, and the various interactions that they will need to have with the process. Thus, each use case is a different view, or projection, of the process. In contrast, the PDL approach presents the entire process itself, indicating which steps are to be executed by different users. Thus, a PDL definition depicts the entire process, including, in particular, steps that might be performed by software systems, and might therefore be transparent to any human user. Use cases, on the other hand, do not describe how the system is going to work internally–only what it is intended to deliver. But it is important to clarify that what is presented to users is actually done correctly. Consequently, PDL definitions focus on what is actually done, and how, by whom, when, which are all necessary in order to support demonstrations that the outcome of a process is done correctly.[2]

A particularly serious drawback of the use case approach is that each of the potentially myriad variations in the path within a process must be described as a separate use case, often leading to a voluminous set of process definitions for one completely described process. For example,

in a license renewal process, one possible path might be that the process halts unsuccessfully because some information on the applicant is missing. In another situation, all information might be available and the renewal process completes successfully. In a use case setting, each of these alternatives would need to be written up as separate use cases. This creates a heightened risk that similar variations in use might wind up being described in ways that have needless differences. PDL definitions avoid this difficulty by allowing the analyst to specify all the alternative flows or paths through a process in the one process definition, rather than requiring separate definitions to be written for each alternative process scenario.

In 2001-2002, the three first authors led a team that used Little-JIL as a tool to analyze existing use case descriptions that were guiding Mass.gov developers and to report any identified process defects (Sondheimer et al., 2002). Once the processes were well defined, they were to be implemented by a state contractor. We analyzed nineteen different license renewal processes written in use case format (listed in Table 1). While each of these cases exhibited differences, many followed a generic structure, which suggested the possibility of reuse. For brevity and demonstration sake, we provide a Little-JIL representation of some of this generic structure (Figure 2).

The top step of this process, "Renew License," consists of the sequential (by virtue of the right arrow icon) execution of the sub-steps, "Identify Applicant," "Validate Applicant Context," "Process License Renewal," "Calculate Payment," "Submit Payment" and "DOI Review of Documentation," all of which are connected to "Renew License" by lines emanating from the right arrow. "Identify Applicant" consists in turn of one sub-step, "Authenticate User." Most other steps have substep decompositions. For brevity, they are not shown, but exist as separate step diagrams. Little-JIL also supports the definition of data flow between steps, by attaching annotations to the lines connecting the steps. Diamonds imbedded in the middle of these lines contain arrows depicting the direction of data flow. For example, the line connecting "Authenticate User" to its parent step "Identify Applicant" has an upward arrow in its associated diamond, indicating that it provides data to the parent. For clarity, further annotation is provided to the left of this line that describes the data flowing upward as "User ID." A similar data flow annotation is attached to the line connecting parent step "Renew License" to its second sub-step "Validate Applicant Context." Here the data "User ID" flows from parent to child, and as a result of that sub-

TABLE 1. The Nineteen License Renewal Use Cases Reviewed for the Commonwealth of Massachusetts Mass.gov Project

Board of Registration in Medicine (4 cases)

- Full medical license renewal
- Lapsed medical license renewal
- Limited medical license renewal
- Acupuncture license renewal

Alcoholic Beverage Control Commission (14 cases)

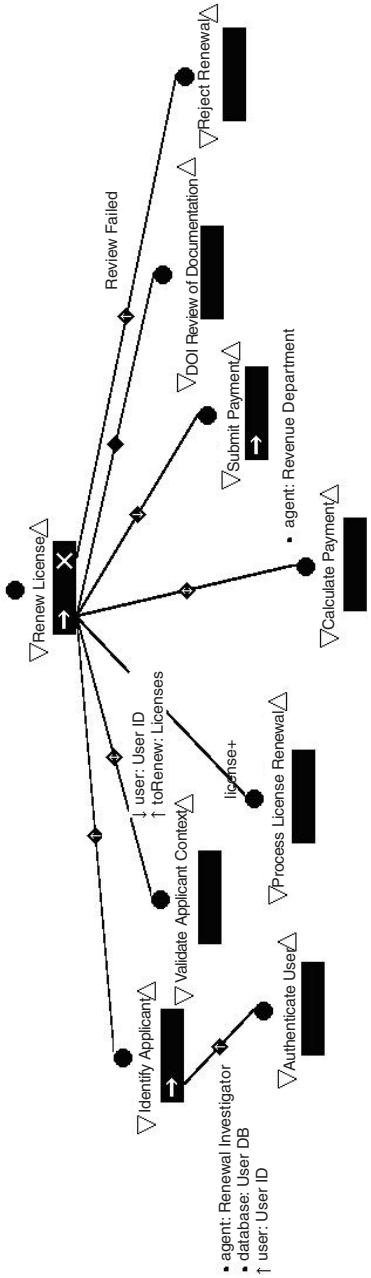
- Wholesaler, manufacturer, or commercial license renewal
- Salesman and transportation for salesman license renewal
- Storage license renewal
- Transportation and delivery license renewal
- Airline sale, airline transportation, railroad sale, and railroad transportation license renewal
- Broker license renewal
- Ship sale license renewal (2 types)
- Express carrier and caterer transportation license renewal
- Public warehouse or bonded warehouse license renewal
- Ship chandler license renewal
- Retail license renewal–local town board
- Retail license renewal–licensee
- Out of state supplier certificate of compliance renewal

Auctioneer License Renewal (1 case)

step the data "Licenses" flow back from child to parent. Other annotations are elided to save space and clutter for purposes of this discussion.

This diagram also depicts the usage of resources. In particular, the line between "Identify Applicant" and "Authenticate User" is also annotated (to the left) with "database: User DB," indicating that the "Authenticate User" step requires the database User DB as a resource in order to be successful. The annotation also defines that the agent responsible for executing this step must be a "Renewal Investigator." The agents responsible for most other steps are not shown, again for brevity, although note that the agent for the "Calculate Payment" step is defined to be "Revenue Department." Thus, this process definition is precise and articulate in indicating which agents are responsible for which steps. Some agents are people, some are organizations, and some could be automated devices such as software programs. Little-JIL supports specification of such mixed collections of agents.

FIGURE 2. A Generic License Renewal Process



Finally, note that this Little-JIL process defines one exception, "Review Failed," and how it is handled. The step "Reject Renewal" is the process for handling this exception by virtue of its being attached to the "Renew License" parent step by a line emanating from the X icon at the right of the step bar. In Little-JIL this means that if a "Review Failed" exception occurs in any of the sub-steps of "Renew License" it is to be handled by executing the "Reject Renewal" sub-step. Possible places where such an exception might take place include the "Authenticate User" sub-step and the "DOI Review of Documentation" sub-step. A full Little-JIL definition would annotate all such steps to indicate that they are possible sites for the occurrence of these exceptions. The upward arrow imbedded in the line connecting "Reject Renewal" to its parent "Renew License" indicates that the execution of the entire "Renew License" step is to be aborted once "Reject Renewal" has completed. For brevity, other sub-step processes are not shown, but readers interested in Little-JIL specifics are encouraged to read Wise (1998) or Cobleigh, Clarke and Osterweil (2000).

RESULTS

We argued earlier that two benefits of PDLs over natural language representation are analyzability and reuse. We developed elaborations and variations of the generic structure in Figure 2 for the nineteen license renewal variants listed in Table 1. These PDL implementations coupled with human-based analysis (as opposed to computer-based analysis, discussed further below in the "Concluding Comments" section) resulted in the identification of twelve process irregularities or defects in the natural language documentation. These failings, if left undetected, could have led to the development of online systems whose characteristics and behaviors would be unpredictable and less trustworthy. We also identified three reuse opportunities. A short summary of some of our specific findings follows (for more detail, see Sondheimer et al., 2002), and we provide some example comparisons between the use case descriptions and PDL representations.

We identified five actor inconsistencies in the natural language descriptions of the nineteen license renewal processes, which occurred because looser agent specification in the natural language descriptions allowed for the unintended omission of actors. Let us provide an example. In the "Process Salesman and Transportation for Salesman License Renewal" use case description, there is a section where the analyst lists

relevant "actors." Then, there are paragraphs written on the basic flow of the process, and "alternative flows" which include seven pages of written description. Embedded in two sub-steps in these alternative flow sections were references to the employer as an actor, and yet the employer was not defined in the use case as an actor. The formal semantics Little-JIL provided were effective in helping the human inspectors to identify these problems. The precise nature of Little-JIL required agent specifications at each step. This prevented the accidental omission of these agents, and focused attention on specifying them consistently.

We identified seven errors in the natural language descriptions of the nineteen processes that were caused primarily by lack of information about the flow of artifacts between steps. For example, in the process of translating the "Process Auctioneer License Renewal" use case into Little-JIL, we attempted to include all of the parameter flow information in the Little-JIL representation. While the data required for steps in the use cases are not directly stated with the step descriptions, deducing most of the relationships between the steps and the Data Source Field List (a section of the use case) was not difficult. However, as we attempted to elaborate all of the information requirements for all of the steps, we discovered that not all of it was specified in the use case document. There were several cases in which data were implicitly referred to by the step description, but the data were not present in the Data Source Field List. For example, one of the exception flows documented in the case indicated that the application would be "flagged" to allow for follow up on bond information. This implies that some datum will be stored somewhere indicating this flag. There was no such datum declared in the use case Data Source Field List–an omission. Errors such as this, if left undetected, would result in the online system not working as intended. The use of the PDL helped discover this omission.

We observed other irregularities, which, though related, were slightly more complex. In another exception flow for this same process, the use case indicated that the bond expiration to be checked can be found on the original application and may have been updated if a letter was received from the bond company. However, the use case does not provide a datum for the expiration date, the bond, or the original application. The use case makes no provision for obtaining this needed information.

There were also cases in which the use case failed to clarify what information was needed to complete a step. For example, in the Auctioneer Renewal process, the Basic Flow section did not specify which data needed to be saved at the end of the process. In short, the precision of the Little-JIL PDL forces the analyst to clearly define what input artifacts

are required for the completion of each step, what output artifacts each step produces, and how the artifacts flow among steps. This additional rigor helped us detect artifact flow inconsistencies that went unnoticed in the natural language articulation of these processes and once again, if these were left undetected, the resulting e-government system would not be fully functional.

At this juncture, the reader may be wondering why these errors were identified using the PDL and not through the use of the use case descriptions? Or, could these have been simply poorly written use cases? We will address these issues in the next section, after we discuss our other results–identification of additional reuse opportunities.

In addition to the detection of process description defects, the use of the PDL improved our ability to recognize commonalities or reuse opportunities between the nineteen license renewal variants. When reviewing the natural language descriptions of processes, finding commonalities and abstracting the commonalities to a higher level was inefficient and difficult. Missing these commonalities could result in errors and inconsistencies between processes and missed opportunities to apply the same logic in two or more license renewal variants. The abstraction capability in the Little-JIL PDL was particularly helpful in identifying these commonalities. It assisted the human analysts to identify logical clusters of related process steps and then group these steps together as an abstraction, denoted by a higher-level step. This higher-level abstraction made it easier to identify where else this abstraction could be used.

Table 2 provides one example of the reuse opportunities we discovered in the use case text. Note that the natural language descriptions for the two separate license renewal processes existed in sixteen pages of natural language text. It would not be very easy to discover these similarities in that representation of the processes. In the Little-JIL description, this step was represented by a step name "Gather Applicant Information" for both processes. This level of step aggregation or abstraction made it easy for our team of analysts (inspectors) to identify these reusable components between applications.

DISCUSSION

Our Mass.gov experience provides empirical support for our main hypothesis: *The rigor and precision of a PDL helps ensure that processes are defined more thoroughly and precisely, facilitating the identification of errors and inconsistencies in natural language-based*

TABLE 2. An Example of a Reuse Finding (Use case text provided by Massachusetts Office of Consumer Affairs)

Step 4 of "Process Retail License Renewal–Licensee" (use case natural language)

The user is asked to complete the following Applicant Information. Note: This information appears in the signature area of the current application.

a. Name of signatory
b. Date
c. Telephone number
d. Social Security number (if individual) OR federal ID number (if corporation)

The user completes the information in the area provided.

Step 9 of "Process Out of State Certificate of Compliance Renewal"

The user is asked to complete the following Applicant Information. Note: This information appears in the signature area of the current application.

a. Social Security number (if individual) or federal ID number (if corporation)
b. Individual's name (if individual)
c. Corporate officer name (if corporation)
d. Business mailing address
e. Date

The user completes the information in the area provided.

process definitions. There are several reasons why we were able to detect problems in our inspections of Mass.gov use case descriptions using Little-JIL and why these were difficult to detect in the use case descriptions.

First, the PDL is particularly amenable to rigorous and definitive analysis or inspection. Little-JIL's carefully defined syntax badges (including pre- and postrequisites, control flows, interfaces, exceptions, reactions and handlers) provide precision to the documentation of processes. Moreover, the PDL requires the specification of agents and exceptions as part of every step, which therefore encouraged us to think about just who was responsible for executing each step, and what might go wrong at each step. In short, the formal requirements of the PDL forced the analysts to think more carefully about components such as agents and exceptions. Natural language-based tools such as use cases did not have such formal requirements and therefore it is all too easy to overlook a missing agent. It was not uncommon for us to find that process components like agents, exceptions, and others listed above had not previously been considered in the natural language descriptions of

the processes, not because the use cases were poorly written, but because the use case methodology does not provide nagging requirements to remind the analyst to include these details. The PDL descriptions were more complete, more accurate, and better understood than their natural language counterparts.

Second, as we have stated, use cases focus on the end-user view of the world, and do not pay as much attention to how things get done in the process, and parts of the process that are not visible to the user. This can lead to arbitrary choices of agents to perform activities that are not clearly visible to the user. The PDL formalism requires that all activities be specified and thought-through. Activities (especially "hidden" activities) that are common to more than one use case will, in the PDL definition, receive one agent, thereby avoiding these difficulties found in use case descriptions.

Third, Little-JIL's visual or graphical nature helped to construct a clear, readable, and understandable structure for processes we analyzed. The hierarchical structure of the language helped clarify which exceptions are handled, at which places and in which ways. This makes it more straightforward for human analysts to review such process definitions to assure that necessary exceptional conditions are handled properly. While it takes some initial effort to learn the language semantics, our experience has been that the level of effort is modest, and once the analyst does this he or she can quite readily understand the structure of processes and make indicated adjustments quickly and surely.

Fourth, because the language is broadly applicable and expressive, we were able to easily apply it to all nineteen license renewal processes. In fact, the components of Little-JIL are capable of describing most imaginable processes (online or offline).

In addition to the analyzability property, our experiences with the Commonwealth of Massachusetts analysis supported our contention that PDLs are particularly effective in supporting reuse. Here again, the support of abstraction through the hierarchical structure of the language is most useful and important. During the activities directed towards identification of the principal parts of the license renewal process, we were struck by the fact that activities such as the checking of credentials and application details appeared to take place at different times. Having once defined these activities as process steps, we questioned whether their occurrences at subsequent times were no more than repetitions of the earlier activities, just in different contexts. This led the Little-JIL analysts to make significant reuse of process steps. We do not mean to imply that analysts working with natural language descriptions could not

discover reuse capabilities. But, in this experience, we identified several instances where the reuse opportunities were not recognized during the use case development process. The graphical and abstract descriptions of the PDL made it easier for us to identify these opportunities.

This reuse capacity of Little-JIL, and PDLs more generally, will have efficiency benefits in e-government applications and in the implementation of government processes in general. For example, U.S. states process many different types of licenses, but the renewal structure is often similar between them. Analysts can start with the generic structure of a process that over time has proven itself to be secure; and, if documented in a PDL, analysts could then easily copy the process structure and apply it in other relevant contexts.

Until now, our discussion about the utility of and our experience with PDLs has been generally positive. But readers might be asking several questions: First, why couldn't the errors detected through the use of PDLs also be detected using more traditional methods, such as through rigorous testing? Second, what are the downsides to using a PDL? Let us address both of these questions.

It is possible that the errors we caught could have been detected through a close examination of the natural language or in future testing of the e-government application. However, in a classical observation, Dijkstra (1972) once said that testing can demonstrate the presence of errors in computer programs, but is hopelessly inadequate for showing the absence of errors. Indeed, the absence of errors could only be shown by executing, and studying, a completely exhaustive set of test cases for a piece of software. But even small, straightforward pieces of software can have myriads of possible execution paths, when considering (as is necessary) all possible logic flows and input values. Thus, totally exhaustive testing is not a feasible possibility. This explains why so many errors in application software are encountered by end users. Consequently, people in the disciplines of computer science and software engineering have advocated the use of "static analyzers" to complement testing approaches, and to demonstrate the absence of certain classes of errors. This capability for demonstrating the absence of errors seems to us to be of great importance in assuring the quality and reliability of e-government processes. Static analyzers rely on documented models of processes they are to analyze, and their accuracy in diagnosing errors (or proving their absence) is dependent on the accuracy, precision, correctness, and completeness of the models upon which they work. Thus, natural language is an unacceptable medium for the definition of models that are subjected to static anal-

ysis. PDLs open up the possibility of detecting defects in processes without the need for exhaustive testing, which is in many cases practically impossible anyway.

Let us turn to the second question that readers might be asking: What are the downsides of Little-JIL and PDLs, in general? Like many languages, Little-JIL is relatively complex and powerful, and consequently it requires some substantial upfront costs and effort to learn how to use it effectively. Our experience with it shows that the subtle attributes of the language make it easy for the analyst to make mistakes, particularly in the early stages of using it. In addition, the Little-JIL end-user interface software is still primitive and hard to use. This leads us to the most important limitation of PDLs in general: the field is still in its infancy, and much of the available PDL work is still in prototype form. Beyond what is reported in this paper, there is growing evidence that PDLs can be quite effective in defining processes–especially complicated ones (Cass and Osterweil, 2004; Ellison et al., 2004). But much remains to be done to determine just what these process description languages should look like and what supporting end-user tools are needed to make them more intuitive and easy to use.

CONCLUDING COMMENTS AND A LOOK TO THE FUTURE

E-government developers need to clearly and precisely define the processes they are automating. In this paper we argued that there is a spectrum of process definition approaches, from more readily comprehensible (by humans, not computers) approaches such as data flow diagrams and workflow descriptions to more formal and precise approaches we call Process Definition Languages. We argued that the qualities PDLs provide–rigor, precision and clarity through the use of graphics and abstraction–improve the chances that human analysts or inspectors will more thoroughly articulate a process or identify errors or omissions in process logic than when processes are articulated in natural language. We presented an example of one such PDL called "Little-JIL" and described how this PDL helped us identify errors and omissions in natural language descriptions of license renewal processes during the development of the Commonwealth of Massachusetts' e-government portal. The PDL application also helped us identify opportunities for process reuse that were not originally detected in standard use case descriptions of the same processes.

But the analytic benefits we have shown here related to human inspectors using PDLs are only scratching the surface compared to the potential benefits PDLs could provide in the future as these technologies advance. In the Mass.gov case, our team of human analysts discovered errors, omissions and reuse opportunities in the use case descriptions because the Little-JIL PDL forced them to be rigorous and precise in their definition of the processes. Still, human inspection cannot prove definitively (as in a mathematical proof) that a process definition is error free, which is where the future of PDLs and process description in e-government, and in public administration in general, lies.

Software engineers have developed automated (computer software) approaches to analyzing software logic for defects. Osterweil (1987) makes the argument that organizational processes are one type of software. They exhibit logic, structure, dataflows, and modules (subtasks) just like computer software, but tend to reside in the minds of humans rather than in the hard disks of computers. Consequently, automated analysis techniques developed in the field of software engineering to detect software defects could be applied to the analysis of process descriptions. This is where the field of PDLs is now moving. It has important implications for the quality of future e-government processes, and has broader implications for the analysis of manual processes that need high levels of assurance that they are defect-free (such as processes related to homeland security). But just how might future analysis of public administration processes be done?

In the field of software engineering, there are two main types of automated software analysis approaches, static and dynamic, that can be applied to the analysis of online or offline processes (or a combination of the two) described in a PDL. The most elementary automated static analyzers are rule-checkers, namely computer programs that can detect defects predefined in process logic. Generic rules to be checked might include, for example, the verification that all inputs needed for a step in a process are defined in some previous step, or the identification of step outputs (e.g., a report) that are never used in later stages of the process. Analyzers that are effective in detecting these kinds of situations may identify processes that are incomplete, flawed, or produce unneeded and unwanted process outputs. Thus, rule-checking static analysis could contribute to increased e-government and manual process trustworthiness through defect removal or identification of products that are never again used.

Another area where rule-checking could be applied is in the comparison of processes that are intended to mirror one another. This is an important element of e-government application development, where in many or most cases online processes should be identical to the over-the-counter version of that same transaction process. Indeed, in most settings, because of digital divide issues, e-government will not entirely replace over-the-counter processing. Therefore, an important concern of public administrators will be whether online and offline processes follow the same logic. If online and offline processes are articulated precisely in a PDL, the opportunity exists for the analyst to apply a static analysis program that would look to ensure that the two processes are indeed identical or to reveal where differences exist. These kinds of "difference" tools are common in the field of software development. There is no reason why they cannot be applied to the study of organizational processes.

Finite state verification (FSV) is a more sophisticated automated static analysis approach. FSV provides analysts with the ability to define customized rules to be checked (e.g., defining a rule to check for prior authorization before moving to the next process step). It also provides automated tools to scan process sequences for violations of these custom properties. FSV tools can therefore locate troublesome execution sequences, or, more importantly, prove that a process definition is completely free of such sequences.

While automated static checkers provide powerful validations of processes that should greatly enhance our ability to create error-free e-government applications, some stakeholder doubt could remain, particularly in processes handling sensitive data or ones needing high levels of security. Dynamic analysis can help here, as a technique for monitoring ongoing process execution. PDL process representations allow the analyst to specify where and when monitoring information is to be generated. During process execution, PDLs can produce audit-trail information for review by humans or computer programs external to the process. These audits can be conducted either in real-time, or post-execution, to detect rule violations. Real-time detection could lead to on-the-spot correction. Post-facto execution could trigger compensatory efforts. Both would lead to increased quality in e-government applications.

In short, the analytic benefits of PDLs demonstrated in this study provide an initial demonstration of some of the benefits PDLs could offer to public administrators and e-government developers in the future. Many other analytic benefits are coming as PDLs advance. But for public organizations to move to this next level of process analysis

and error detection, the first step is to move government organizations toward more formal and rigorous process definitions of e-government (and offline) processes. This paper introduced PDLs to public administration theorists, practitioners, e-government developers, and other stakeholders to encourage their future use and deployment.

Received: 05/15/04
Revised: 09/19/04
Accepted: 11/05/04

ACKNOWLEDGMENTS

This material is based upon work supported by funds from the Commonwealth of Massachusetts' Information Technology Division and the National Science Foundation under Grant No. EIA-223599. The authors would like to thank Val Asbedian, Tim Healy and Tom Price for supporting and helping supervise the work from the Government side, Matt Billmers and Joel Sieh for their modeling efforts, and Sandy Wise, the lead developer of Little-JIL, for his continuing support and production of figures. The authors also thank three anonymous reviewers for extremely helpful comments on an earlier draft.

Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the Commonwealth of Massachusetts or the National Science Foundation.

NOTES

1. In the interest of brevity, we do not present a complete example of a use case. Many of the use cases we analyzed were over 10 pages in length. Describing the use case approach would have added significantly to the length of this paper. Interested readers on the use case approach are encouraged to review Cockburn (2001) or Schneider and Winters (2001).

2. It is still true that data flow diagrams and workflows also try to get at this issue of how results are to be obtained. But we have already pointed out that, insofar as they offer the same semantics as process languages, we will consider them to be process languages. Most data flow diagrams and workflows, however, have weaker semantics that do not enable them to capture important details and nuances that can have a major impact upon whether or not a process is done correctly–even in non-nominal cases.

REFERENCES

Aho, A. V., Hopcroft, J. E. & Ullman, J. D. (1983). *Data structures and algorithms*. Reading, MA: Addison-Wesley.

Bandinelli, S. & Fuggetta, A. (1993). Computational reflection in software process modeling: the SLANG Approach. *Proceedings of the Fifteenth International Conference on Software Engineering*. Baltimore, MD. 144-154.

Bandinelli, S., Fuggetta, A. & Grigolli, S. (1993). *Process modeling in-the-large with SLANG. Proceedings of the Second International Conference on the Software Process*. Berlin, Germany: IEEE Computer Society Press.

Belkhatir, N., Estublier, J. & Walcelio, M. L. (1991). Adele 2: A support to large software development process. *Proceedings of the First International Conference on the Software Process*. Redondo Beach, CA. IEEE, New York, 159-170.

Belkhatir, N., Estublier, J. & Walcelio, M. L. (1993). Software process model and workspace control in the Adele system. *Second International Conference on the Software Process*. New York, 75-83.

Belkhatir, N., Estublier, J., & Walcelio, M. L. (1994). ADELE-TEMPO: An environment to support process modeling and enactment. In A. Finkelstein, J. Kramer and B. Nuseibeh (Eds.) *Software process modeling and technology* (pp. 187-222). New York, NY: John Wiley & Sons Inc.

Cass, A. G. & Osterweil, L. J. (2000). Design guidance through the controlled application of constraints. *Proceedings of the 10th International Workshop in Software Specification and Design (IWSSD 10)*, 195-199. Retrieved August 8, 2004, from *http://laser.cs.umass.edu/techreports/00-67.pdf*.

Cass, A. G. & Osterweil, L. J. (2004). Process support to improve novice software designer performance. Technical Report UM-CS-2004-080. University of Massachusetts: Department of Computer Science.

Cobleigh, J. M., Clarke, L. A. & Osterweil, L. J. (2000). Verifying properties of process definitions. *Proceedings of the ACM Sigsoft 2000 International Symposium on Software Testing and Analysis (ISSTA) 2000,* Portland, OR, 96-101. Retrieved August 8, 2004, from *http://laser.cs.umass.edu/techreports/00-65.pdf*.

Cockburn, A. (2001). *Writing effective use cases*. Reading, MA: Addison-Wesley.

Dennis, A., Wixom, B. H. & Tegarden, D. (2002). *Systems analysis and design: An object-oriented approach with UML*. Hoboken, NJ: John Wiley and Sons.

Dijkstra, E. W. (1972). The humble programmer. *Communications of the ACM,* 15 10, 859-866.

Ellison, A. M., Osterweil, L. J., Hadley, J. L., Wise, A., Boose, E., Clarke, L., Foster, D. R., Hanson, A., Jensen, D., Kuzeja, P., Riseman, E. & Schultz, H. (2004). An analytic web to support the analysis and synthesis of ecological data. Technical Report UM-CS-2004-080. University of Massachusetts: Department of Computer Science.

Estublier, J. A. (1985). Configuration manager: The Adele data base of programs. *Workshop on Software Engineering Environments for Programming-in-the-Large*. Harwichport, MA.

Heineman, G. T., Kaiser, G. E., Barghouti, N. S., & Ben-Shaul, I. Z. (1992). Rule chaining in Marvel: Dynamic binding of parameters. *IEEE Expert*. 7 6, 26-32.

Ho, A. T. (2002). Reinventing local governments and the e-government initiative. *Public administration review,* 62 4, 434-444.

Hoffman, D. M. & Weiss, D. M. (Eds.). (2001). *Software fundamentals: Collected papers by David L. Parnas*. Boston, MA: Addison Wesley.

Kaiser, G., Barghouti, N. S., & Sokolsky, M. H. (1990). Experience with process modeling in the MARVEL software development Environment kernel. Proceedings of the *23rd Annual Hawaii International Conference on System Sciences*. 1990. Kona, HI.

La Porte, T. R. & Thomas, C. W. (1995). Regulatory compliance and the ethos of quality enhancement: Surprises in nuclear power plant operations. *Journal of Public Administration Research and Theory,* 5, 109-137.

Moon, M. J. (2002). The evolution of e-government among municipalities: Rhetoric or reality? *Public Administration Review,* 62 4, 424-433.

Osterweil, L. J. (1987). Software processes are software too. *Proceedings of the 9th International Conference on Software Engineering,* 2-13.

Schneider, G. & Winters, J. (2001). *Applying use cases: A practical guide*. Addison-Wesley: New York.

Seville European Council. (2002). eEurope 2005: An information society for all. An action plan adopted by the Seville european council. Retrieved May 18, 2004, from http://europa.eu.int/information_society/eeurope/news_library/eeurope2005/index_en.htm.

Sondheimer, N. K., Osterweil, L. J., Schweik, C. M., Billmers, M., Canavan, D., Kelly, A., Lee-Davis, C., Li, C. & Sieh, J. (2002). *Online license renewal analysis: Process modeling and state practice review final report*. Amherst, MA: Electronic Enterprise Institute and the Center for Public Policy and Administration.

Szyperski, C. (2002). *Component software: Beyond object-oriented programming*. Addison-Wesley: New York.

Tufte, E. R. (2001). *The visual display of quantitative information*. Cheshire, CT: Graphics Press.

Weber, M. (1946). Bureaucracy. In H. H. Gerth & C. W. Mills (Eds.), *From Max Weber: Essays in sociology* (196-244). New York: Oxford University Press.

West, D. (2002a). *State and federal e-government in the United States*, *2002*. Retrieved January 24, 2004, from *http://www.insidepolitics.org/Egovt02us.html*.

West, D. (2002b). *Urban e-government, 2002.* Retrieved from *http://www.insidepolitics.org/egovt02city.html*.

Wise, A. (1998). Little-JIL 1.0 Language Report. Amherst, MA: University of Massachusetts, Department of Computer Science.