Balboa: A Framework for Event-Based Process Data Analysis

Jonathan E. Cook

Department of Computer Science
New Mexico State University
Las Cruces, NM 88003 USA

jcook@cs.nmsu.edu

Alexander L. Wolf

Department of Computer Science
University of Colorado
Boulder, CO 80309 USA

alw@cs.colorado.edu

University of Colorado
Department of Computer Science
Technical Report CU-CS-851-98 Feb. 1998

New Mexico State University
Department of Computer Science
Technical Report NMSU-CSTR-9809 Apr. 1998

A version of this report to appear in
Proceedings of the 5th International Conference on the Software Process

ABSTRACT

Software process research has suffered from a lack of focussed data analysis techniques and tools. Part of the problem is the ad hoc and heterogeneous nature of the data, as well as the methods of collecting those data. While collection methods must be specific to their data source, analysis tools should be shielded from specific data formats and idiosyncrasies.

We have built Balboa *as a bridge between the data collection and the analysis tools, facilitating the gathering and management of event data, and simplifying the construction of tools to analyze the data.* Balboa *is a framework that provides to collection methods a flexible data registration mechanism, and provides to tools a consistent set of data manipulation, management, and access services. It unites the variety of collection mechanisms with the variety of tools, thus paving the way for more extensive application of process improvement techniques based on data analysis.*

This work was supported in part by the National Science Foundation under grant CCR-93-02739 and by the Air Force Material Command, Rome Laboratory, and the Defense Advanced Research Projects Agency under Contract Number F30602-94-C-0253. The content of the information does not necessarily reflect the position or the policy of the Government and no official endorsement should be inferred.

1 Introduction

Software process engineering has an advantage over other disciplines in that much of its activity takes place on computers. Thus, it is more amenable to reducing the effort needed for process tracking and analysis, key practices in having a continuously improving process and in the consistent on-time delivery of reliable, profitable software. Indeed, in the past several years, there have been efforts to collect process data and analyze it to improve the process. This work, so far, has seen the creation of single tools that access process data in an ad hoc manner. Several methods for collecting process data have been proposed and constructed (e.g., [5, 9, 27, 29]). However, there has not been a significant effort to propose a coherent framework in which to perform analysis of process data.

Lott [23] gives an extensive summary of process support and measurement support in seventeen software engineering environments. Most process execution or guidance systems have little or no measurement support, and those that do tend to concentrate heavily on product measures to infer process characteristics; a few make use of aggregate process statistics, such as cumulative personnel effort and computer usage time. In most systems, the type of data and how it is used is fixed and, in general, no effort was made to allow extensions to the data collection and analysis methods.

This paper describes Balboa, a framework that contributes data management and analysis support to the field of software process. From the data access perspective, Balboa isolates the tools from the variety of data formats and provides a consistent access method to all of the data, reducing the effort needed to create an analysis tool usable with multiple data formats. From the data management perspective, Balboa provides for the management of data, eliminating the need to provide such (redundant) facilities in each and every tool, and for each and every data format.

Balboa provides this support mainly for the use of event-based data. We concentrate on event data because it supports a wide variety of behavioral and temporal analyses. Balboa provides a foundation from which to more easily construct tools, and offers facilities for managing event data and for specifying descriptive meta-data. Balboa is a freely available system, along with several analysis tools that we have built in the course of our software process research.[1]

An existing system that comes closest to claiming the position of a data analysis framework is Amadeus [27]. It is flexible in the specification of what data to collect and what to do with the data. It is positioned as a supporting unit to a process-guidance system, so it is meant to integrate other tools. However, it is essentially a system for registering process events and triggering actions on the occurrence of those events. Thus it cannot be considered a complete framework for collecting, managing, and providing event data to a variety of analysis tools. Similarly, the TAME project [4] is centered around an integrated environment, but is not intended to manage data collected outside of itself and to integrate external analysis tools.

This paper has the following organization. Section 2 describes the nature of event data, its strengths and weaknesses, and how it can be collected and used. Section 3 presents the Balboa framework and its features. Section 4 then describes the use of Balboa through a scenario based on our experiences in a process analysis study. Finally, Section 5 concludes with some insights and ideas for future work on Balboa.

[1] http://www.cs.colorado.edu/serl/process/Balboa

2 Event Data

Software processes can be viewed as real-time, concurrent systems. They are real-time because they have constraints, whether hard or soft, on when certain points in the process must be reached (delivery of a final product on time being the most important one!). They are also concurrent because they have multiple agents, both people and machines, performing activities that overlap in time and are independent.

Analysis of real-time, concurrent systems has long used event data to characterize executions of a system [22, 30]. Event data have been used for checking against specifications [1], for debugging [6, 15], and for behavioral analysis [22]. Temporal and interval logics are also used to perform real-time correctness analysis based on the time between event occurrences [26]. Because of the large body of work applying analysis of event data to systems similar to software processes, we have created the BALBOA framework to allow tools easy and consistent access to event data.

Following Wolf and Rosenblum [29], we define an *event* as an identifiable, instantaneous action, such as invoking a development tool or deciding upon the next activity to be performed. For purposes of maintaining information about an action, events are typed and can have attributes, such as the time the event occurred, the outcome of the action, the agent that generated the event, and the artifact(s) involved in the event.

Because events are instantaneous, an activity spanning some period of time is represented by the interval between two or more events. For example, a meeting could be represented by a "begin-meeting" and "end-meeting" event pair. Similarly, a module compilation submitted to a batch queue could be represented by the three events "enter queue", "begin compilation", and "end compilation". The overlapping activities of a process, then, are represented by a sequence of events, which we refer to as an *event stream.* We also refer to the event stream of a completed process as an *event collection.*

An example event might look like:

```
94/08/23 09:32:32 code-checkin pdel.c efudd MR99732
```

where the event type is a code check in (into a version control system), and the attributes are the date and time the event took place, the file that was checked in, the engineer who initiated the event, and the tracking number to which this change is related. Note that we are not constraining the format of the event; the method we use for interpreting an event and determining its type and attribute values is described in Section 3.2.

2.1 Event Data Collection

Techniques for event data analysis clearly depend on an ability to collect event data from an executing process, but we do not address this topic in this paper, since a variety of methods for collecting process execution data have already been devised:

- Basili and Weiss [5] describe a method for manual, forms-based collection of data for use in evaluating and comparing software development methods. Their methods would be a good place to begin in creating a system for manual event data collection.

- Selby et al. [27] built a system, Amadeus, for automated collection and analysis of process metrics. It acts as an event monitor, allowing the triggering of actions based on certain events. Our use would simply collect the events into an event stream.

- Wolf and Rosenblum [29] use a hybrid of manual and automated collection methods to collect event data from a build process.
- Bradac et al. [9], provided the user with a menu-based tool that collects sampled task and state information. This approach helps ease the burden of collecting off-computer, manual events.
- Cook et al. [10] extracted event data from existing historical repositories and logs automatically kept by tools.
- Krishnamurthy and Rosenblum [21] built a system event monitor, Yeast, that monitors events that occur on computer, and reacts to those events. It also provides functionality for reporting any type of off-computer event as well.

Some enactment systems, such as Provence [3], are already centered around monitoring events, and these types of systems would naturally provide event data collection mechanisms. The Endeavors [8] process infrastructure provides an event-based integration and communication infrastructure, which could easily report events to Balboa for collection purposes. Other process enactment systems such as Oz [7] and SPADE [2] are also capable of being instrumented to collect event data. Process integration components, such as ProcessWall [18], are likewise a natural focal point for such instrumentation.

Many software engineering environments, even those that are not process based, can supply event data relatively easily. Message-based systems, such as Field [25], Softbench [16], and ToolTalk [28], provide a natural framework from which to collect data about activities. Efforts to collect events across the Internet (e.g., user interface events [19]) are also compatible with our approach.

Of the existing tools that developers already use, many naturally provide some event data collection as well. For example, all configuration management systems provide a history of the actions taken on each artifact under control, and many organizations have in-house tools, such as problem tracking systems, that track development or maintenance activity. These existing sources, when merged, can provide a comprehensive event history of a process.

Any off-computer data collection will have to have some sort of manual recording component, but some of the systems described above allow for the recording or announcing of this information to the collector. For example, one can use the `announce` mechanism in Yeast to collect data about meeting and phone communication events [21].

We do not address the issue of data integrity in this paper. We assume that prior to the application of analysis tools the data have been examined and, if necessary, cleaned. This is not to say that there is no noise in the data, but that it has been eliminated as much as possible. Eliminating noise is a general concern that begins with the collection methods and continues through possible consistency analyses on the data. This process can be thought of as an analysis tool in its own right, but one that creates a new event stream from an existing one.

3 The Balboa System Framework

Balboa is a client-server framework for tools and data collection services, where a server provides client tools a uniform access interface to heterogeneous event data, and provides a flexible data submission interface to the collection methods. Clients can be distributed across the Internet from the server with which they are communicating. Figure 1 shows a high-level view of the Balboa framework. Three access channels are provided to a Balboa server:

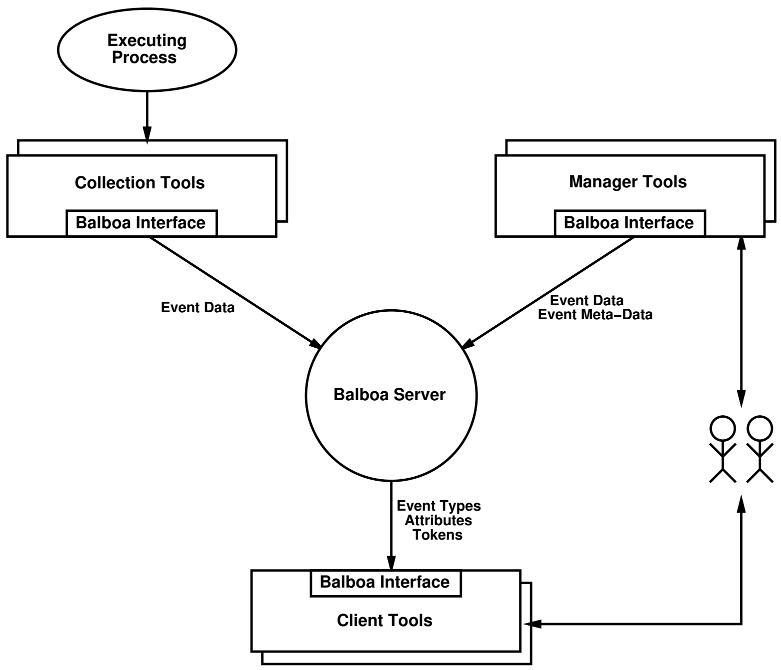

Figure 1: The BALBOA **Framework.**

- A *data collection* interface allows site-specific collection tools to submit collected event data to a server.
- A *data management* interface allows management tools to manage event data and create meta-data for describing the makeup of event data.
- A *client tool* interface allows client analysis tools to access event data in a consistent manner.

These interfaces consist of messages that are sent between the clients and the server. Additionally, BALBOA provides support for the clients in the form of language-specific libraries and APIs for accessing the capabilities of the messaging interface to the server.

As shown in Figure 1, the client tools do not normally access raw event data. Instead, they see *interpreted* data: event types, attributes and values, and tokens (event types mapped to integer identifiers). The manner in which the interpretation takes place is specified by the meta-data that the data management tools allow the user to create, as described in Section 3.2. This interpretation frees the tools from the specific format of each data source, and lets the tool builders concentrate on building the analysis methods rather than on data manipulation.

While BALBOA is concentrated on managing and providing event data to client tools, non-event data is also supported. User-defined process attributes can be registered with a BALBOA server. These attributes are arbitrary name-value pairs that are attached to an event collection. They can be used for simple aggregate process metrics, such as the outcome of the process, the total size of the project, and other such values. This feature is rudimentary and is an area of ongoing expansion.

For security, a simple authorization mechanism exists for controlling locations from which analysis tools can connect to a given BALBOA server. Access control is based on Internet addresses, so that a server may be accessible only, for example, from within the company or group that executes the process.

Finally, Balboa provides a suite of management tools that use the data management interface to provide the fundamental management needs in using Balboa.

3.1 The Messaging Interface

Balboa relies on the reliable Internet protocol (TCP sockets) as the communication layer for the tool interfaces. It thus provides an Internet-level service to client tools. Messages are sent bi-directionally through the server-tool connection. Messages sent from clients are commands to the server, and the server replies with one or more data and/or status messages. In addition to command messages, data messages are also sent to the server in the data collection and management interfaces.

Though the messaging interface is mostly a command-response protocol, there are higher-level protocols that order what messages can be used and when. For example, a command message selecting an event collection must be sent from an analysis tool before any command messages are sent that request events from that collection. The discussion of these protocols is beyond the scope of this paper.

The messaging interface reflects the fact that Balboa supports two kinds of processes and the associated analysis tools that can accompany them:

1. Processes that are simply logging their events for later use by analysis tools, after the process completes.

   The event data may be collected apart from any communication with a Balboa server, and then uploaded to a server after the process completes. Alternatively, it may be sent incrementally to a server. Once the process completes, the data are then a static *event collection* that represents one execution of a particular process. Tools that analyze this type of data are post-mortem analysis tools.

2. Processes that send event data to a Balboa server as the events happen, and which are intend to be analyzed in real time as the processes execute.

   We term these *on-line* processes. Analysis tools that monitor on-going processes can use the data to provide immediate feedback as the process is executing. The data also may be logged by the server to obtain a static collection for later analysis once the process is completed.

These two types of interactions are supported differently by Balboa. The first is supported by the standard view of a client-server relationship—that the server waits for client requests, processes them, and responds. Since the server already has the complete event collection that the analysis tool is using, it merely needs to wait for the tool to request more data, and then respond with those data.

The second type of interaction does not fit into the strict client-server relationship. Because the server is also waiting for data to arrive from the process, a client tool cannot simply request data and expect them to be forthcoming. If this interaction model is adhered to, a client can request the next event, for example, and the server may have to wait extended periods for that event to be produced and sent by the process. This would leave an open connection and outstanding request between the server and the client that could be idle for long periods of time, leading to a very fragile system architecture.

For handling on-line processes and their analysis tools, then, BALBOA requires a tool to register a *callback* address with the server, and then wait and listen for a connection to come from the server at that address. With this, connections between the server and the tools are only made when there are data available to send. This provides a robust mechanism for handling on-line processes.

3.2 Specifying Event Interpretation

As described in Section 2, events collected in an event stream are complex data that have attributes. Tools will want to understand event types and the attribute values rather than just use the raw events. Thus, to make use of the event data in this manner, one must specify a *mapping* of the raw events to specific event types, and to handle attribute matching. These specifications, or *event maps*, are then used by a BALBOA server to interpret the raw event data and extract event type and attribute values from it.

We do not assume that event streams are homogeneous, but rather that it is more likely that they are non-homogeneous, coming from a variety of sources. This was our experience in an industrial study, where events came from a variety of historical repositories [10]. The mechanism to interpret events (i.e., map them into event types and attribute values) is a two-tiered one based on regular expressions and attribute values.

For a given event stream, an ordered set of regular expressions is specified for describing the events in that event stream. Each event is expected to match at least one regular expression. If it matches more than one, the first one it matches is used; if it matches none, then that event cannot be interpreted, which is an error.

The regular expression is piecewise bracketed by "{}" braces in order to recognize attributes of a matching event. Each piece of the regular expression inside a pair of curly braces represents one attribute, and each part of the regular expression outside the braces represents the "white space" between the attributes. The attributes are then named rather than referring to them positionally. To specify a unique event type, a subset of the attributes that the regular expression defines are tagged as being part of the event type. The unique values of the cross product of those tagged attribute types determine the event types in the event stream.

For example, suppose we have an event stream that has the event

```
FILE-WRITE code io.c efudd 15:34:21 96/11/09
```

in it. This event can be mapped by providing the specification

```
{[^ ]*}[ ]*{[^ ]*}[ ]*{[^ ]*}[ ]*{[^ ]*}[ ]*.*
      1=sysop, 2=doctype, 3=doc, 4=user
              type=sysop+doctype
```

where the first line specifies a simple pattern of four attributes not containing spaces and separated by one or more spaces, and ignoring all of the rest (the time and date); the second line names these attributes (`sysop`, `doctype`, `doc`, and `user`, respectively); and the third line specifies the event type as the attributes `sysop` plus `doctype`. An event with `sysop`==`FILE-WRITE` and `doctype`==`design`, for example, would be a different event type from the one shown.

We represent the event type as the concatenation of the tagged attributes (with separating spaces); thus, the event shown above would have an event type of "`FILE-WRITE code`". The attributes `doc` and `user` are variable within an event type. The attribute values are accessible to an analysis tool by name, and might be used, for example, to analyze a particular software module. If the engineer managing this collection decides that the time and date are important attributes, they could change the event map to extract those fields as well.

Our event mapping also supports an encoded version of the event type, called a *token*. A token is an integer encoding of the event type. This encoding reduces a large event stream to a simple, small stream of integers. Accessing a stream in this manner is useful because, for tools that might perform pattern matching or something similar, the need to look at large amounts of data in a simple form is paramount. Rather than have the tool perform the mapping, the BALBOA server provides the mapping for the tool, and also remembers the reverse mapping from token to event type, so that user I/O can be accomplished using the understandable event type, and not the externally meaningless token.

3.3 The Library API for Clients

Tool developers are not required to deal with the raw messaging interface. Rather, language-specific client libraries support the building of tools at a higher level than the messaging layer provides. BALBOA currently supports client tools written in both the C++ and Tcl [24] programming languages, thus allowing both the construction of robust, "polished" tools and the fast exploration of ideas using an interpreted prototyping language.

The language support is tailored to the specific programming language, rather than being consistent across all languages. In C++, for example, an object hierarchy is offered as the API for tools written in that language. A tool can create an *EventServer* object, which when constructed will connect to a BALBOA server and select the specified event collection. Its methods then allow the retrieval of events and event patterns, one at a time. A tool that needs access to the token form of event types can create a *TokenServer* object, which is a subtype of the *EventServer*. It provides the access to events as tokens and the mapping back into event types, as well as the services the *EventServer* object provided.

The support in the Tcl language, since Tcl is not object-oriented, is of the form of a set of functions that support the connection to a server, the selection of an event collection, and retrieval of events. Native Tcl lists and strings are used as the data types for events and meta-data. New versions of Tcl offer modules called *namespaces*, and it would be natural to migrate the BALBOA Tcl support to this better abstraction.

Other languages that have access to a socket interface, such as Java and Perl, could also be easily integrated into the BALBOA framework.

3.4 Management Tools

On the user side, BALBOA also provides four tools for managing data and the user interaction with BALBOA. LAUNCHPAD is a tool that acts as a central execution point for the various manager pieces of BALBOA, and for individual analysis tools. Figure 2 shows a snapshot of LAUNCHPAD. As can be seen, this tool is a simple button-oriented interface to launch the various management and analysis tools of BALBOA. LAUNCHPAD is extensible in that analysis tools can be installed onto it; thus BALBOA helps to manage the tools that use it as well as the data. LAUNCHPAD can specify a BALBOA server as a default that is then inherited by all the tools as they are started up. In Figure 2, the server location is a machine "trutta", and port 2000. This makes the interaction with one BALBOA server transparent across all tools.

Three other tools perform data management functions:

- COLLECTIONMANAGER lets the user create, modify, and delete event collections at a BALBOA server.
- EVENTMAPPER lets the user create and modify event interpretation specifications.

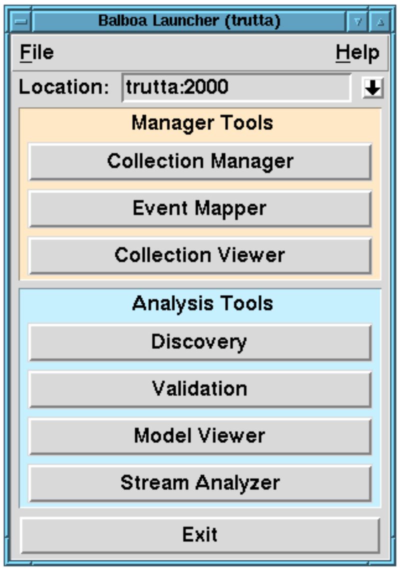

Figure 2: The LaunchPad Tool.

- CollectionViewer lets the user view and browse the event collections, optionally interpreting the events in various ways.

These three tools are discussed in the next section.

4 Using Balboa in Practice

To illustrate Balboa and its managerial tools, we present example uses of Balboa in the study of an industrial software process [10]. Our extensive use of an earlier version of Balboa in the actual study led to several improvements that are reflected in the presentation below. Note that the data we give are a sanitized rendering of the proprietary data used in the study.

The study examined the historical executions of a software maintenance process in a large telecommunications company. The goal was to see if measured process characteristics (especially deviation from a prescribed model) could be correlated with the success or failure of the process.

4.1 Creating and Managing the Data

In looking at historical executions, our data collection consisted of extracting events from databases and logs of various tools, including a version control system, a problem tracking system, and a customer-interaction database. The events were merged from these various sources to form an event stream for each execution of the process. A sample event stream is shown in Figure 3.

In the maintenance process, a customer problem record is created in the customer tracking database when a problem is called in. Once the problem is determined to be related to the software, a modification request is created and assigned to a developer. The code is checked out, modified, and checked back in. After a code inspection, the change is tested, sent to the customer, and, if successful, solves their problem.

```
94/08/23 00:00:00 custd-create IC98234 clos FWA-OC MR991432 C engineerA
94/08/23 00:00:00 custd-abstract IC98234 OS clock synchronization problem
94/08/24 11:03:21 mr-createmymr engineerB mr=MR991432 abst=KDX-II LTIME clock
94/08/24 11:05:04 mr-acceptmr engineerB mr=MR991432 tiu=nontiumr
94/08/24 11:05:07 mr-assign engineerB mr=MR991432 pgmr=engineerB prio=high
94/08/24 11:12:10 mr-asnmymr engineerB mr=MR991432
94/08/24 12:56:54 code-checkin FSgdata.c change engineerB MR991432 2
94/08/24 12:57:05 code-checkin FStimers.c change engineerB MR991432 2
94/08/24 12:57:17 code-checkin FSmachine1.c change engineerB MR991432 2
94/08/24 14:12:12 code-checkin FSmachine1.c change engineerB MR991432 1
94/08/24 14:29:45 code-checkin FSmachine1.c change engineerB MR991432 1
94/08/24 13:30:00 code-inspect 991432 TCDN 1 0 16 0 0 0.0 0.0 1 0 0 0
94/08/24 23:16:43 mr-smit engineerB mr=MR991432
94/08/24 23:16:43 mr-test-plan (engineerB)
94/08/24 23:22:28 mr-smitsys engineerB mr=MR991432
94/08/26 00:00:00 custd-dcreated IC98234 engineerA
94/08/26 00:00:00 custd-inhouse IC98234
94/09/03 00:00:00 custd-custdue IC98234
94/09/16 07:45:22 mr-bwmbuilt engineerC mr=MR991432
94/09/16 07:46:56 mr-acptsys engineerC mr=MR991432
94/09/22 00:00:00 custd-closed IC98234
94/09/22 00:00:00 custd-delivered IC98234
94/09/22 00:00:00 custd-response IC98234
94/09/22 00:00:00 custd-solved IC98234
94/11/05 00:00:00 custd-update IC98234 clos 09-22 Closed - fix verified
```

Figure 3: Sample Process Event Stream.

The merging of data from the various sources was performed by scripts and resulted in a file containing the events for a single execution, such as the one shown in Figure 3. These files were then placed under control of a BALBOA server. Figure 4 shows the collection specification of a particular execution of the process placed under control of a BALBOA server using the tool COLLECTIONMANAGER. This tool allows one to create a new event collection, and to directly import a local file of events into the new collection. The lower display list in the window shows the user-defined attributes of the process—in this case, the outcome was a successful fix, and the size of the change in lines of code was 34. Note that at the top of the tool window is the location of the BALBOA server, in the "Collection Location" field (also the same in figures 5 and 6). This location was inherited from the LAUNCHPAD when the tool was started up.

Along with the collection specifications, an event map must be created with which to interpret the events. Figure 5 shows the tool EVENTMAPPER being used to create the event map. Note that a single event map is shared by all event collections representing executions of the process, so this step is just performed once. The upper portion of the window displays the three regular expressions that match different events in the event streams. The first expression matches the customer database record events. Notice that the "custd" part is within curly braces, so it is part of an attribute (namely, the event type), but that the "IC" part is not, which effectively strips off the two letters from the tracking number (see Figure 3). The second regular expression matches the modification request events, and the third expression matches the rest. Note that the expressions

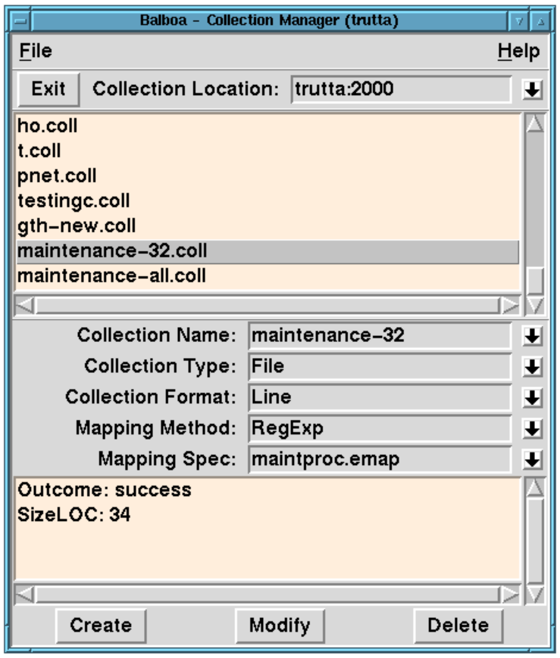

Figure 4: The CollectionManager Tool.

are applied in order, so the most general appears last.

The lower portion of the window shows the attribute names for each of the attributes specified in the selected regular expression (the last one, "Other"). An attribute with a leading "Y" is a component of the event type; in this example, only one attribute makes up the event type, namely the third one.

With the collection registered, and an event map created, we then use the tool CollectionViewer to check our event map for correctness. This is shown in Figure 6. In the figure we are viewing the event stream by event type, so that we can see that the event map is properly interpreting each kind of event. CollectionViewer allows the user to view the raw event, the event type, and all of the event attributes, so that the event map can be completely verified. Once we are assured that the event map is correct, we can then apply analysis tools to the data.

4.2 Analyzing the Data

In this study, we applied two analysis techniques to the event data. Below we describe how the tools that realize these techniques make use of Balboa.

4.2.1 Process Discovery

The first analysis we undertook was applying our technique for automated *process discovery*, which is the use of event data to generate formal models of processes [12, 14]. The goal is to create models with which one can reliably reason about and understand a process, and from which one can construct complete process models.

The process discovery methods analyze streams of event data collected from executing processes

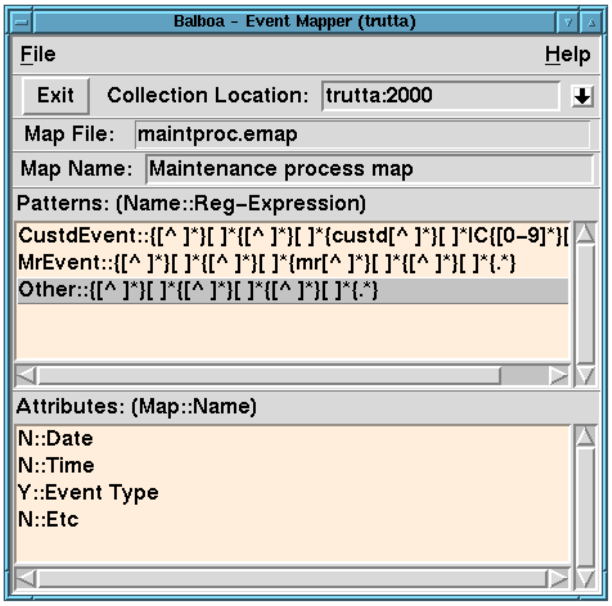

Figure 5: The EventMapper Tool.

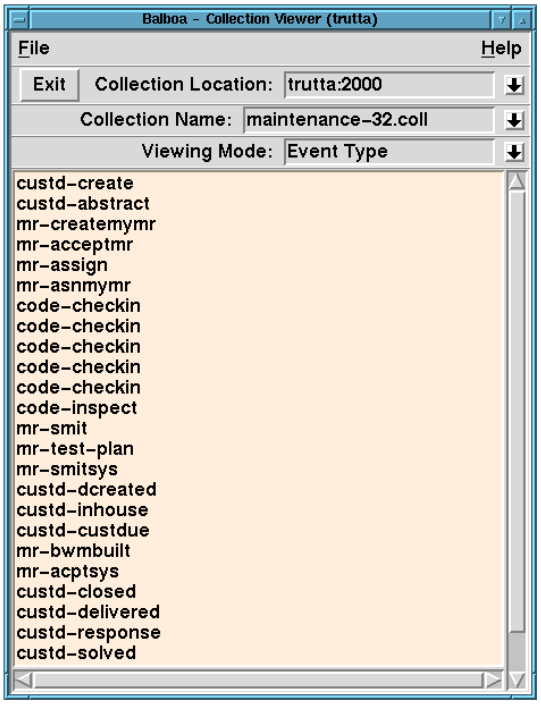

Figure 6: The CollectionViewer Tool.

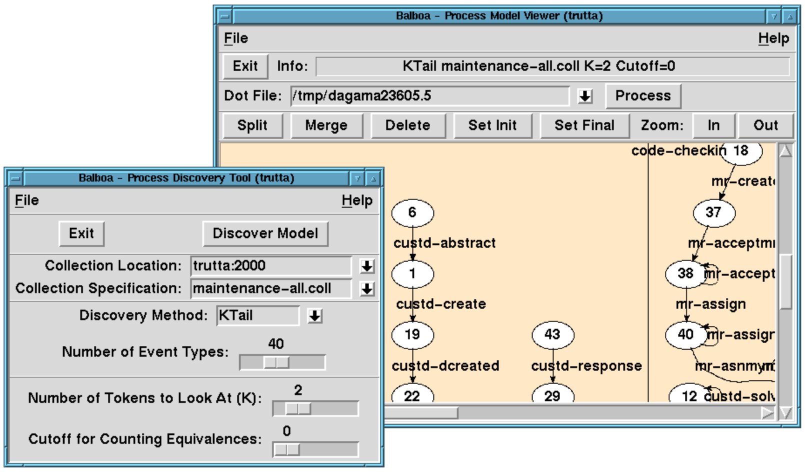

Figure 7: The Process Discovery Tool.

to infer nondeterministic state machine models of behavioral patterns. These patterns represent the major sequencing and iteration of activities within the process, as well as the major decision or branching points within the process.

The process discovery methods are built upon previous work in the field of *grammar inference*, and thus are formulated to analyze sequences of tokens. They benefit from a compact token representation and so use the token interface provided by BALBOA. This freed each analysis tool from the task of performing a token-to-event-type conversion, and allows the tools to be implemented straightforwardly, using the data representation that they expect.

The users of the discovery tools should not know about the token representation, but only about the events and event types in their data. The user interface to the tools therefore use the BALBOA server's mapping of tokens back to event types during user interaction. In addition, the user interface, which is written in Tcl/Tk, employs many of the management functions for convenience, such as retrieving a list of collections to present a selection list from which the user can pick.

The screenshot in Figure 7 shows a model discovered from the event data for the maintenance process under study. The model is a state machine model, with the transitions annotated with the event types that are matched (or produced) when they fire.

4.2.2 Process Validation

The second analysis that we undertook was *process validation*, which measures the correspondence between a formal model of intended process behavior and the actual behavior exhibited by a process [11, 13]. Implicit in our methods is the assumption that a process execution cannot, and indeed should not, exactly follow the model [17]. But the model still represents the idealized process, and relating the actual process executions to the model is necessary in order to understand what is really happening.

For the validation analysis techniques, accessing the event data from a BALBOA server as mapped event types is the method of choice. This is because a model is formulated based on which event type is produced at each step in the model.

In future validation techniques, we envision the attributes of an event playing an important role.

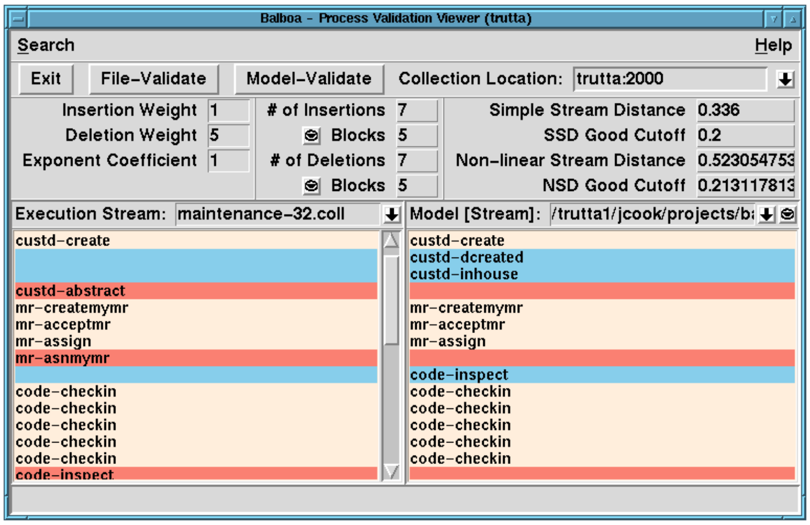

Figure 8: The Process Validation Tool.

For example, making sure the correct document was checked in by the correct person is important, and our current tools do not look for this. For such an extension, the accessing of event attributes by name, provided by BALBOA, will make this enhancement very straightforward.

Figure 8 shows a screen shot from our validation tool. The tool has calculated a measure of deviation for a model and execution-data pair. It displays the expected model stream and the actual event stream side-by-side, allowing interactive browsing of their highlighted differences, and shows the calculated, quantitative deviation metrics in the upper portion of the window.

5 Conclusion

In this paper we presented BALBOA, a framework for consistently managing, interpreting, and serving process event data to analysis tools. The advantages it gives derive from the decoupling of both the data collection and the tool construction from the format and access methods of the data. This separation of tools from data format and management facilitates the construction of tools and allows them to access data from a wider variety of sources. The usefulness of BALBOA has been demonstrated through the construction of process discovery and validation tools, and its use in an industrial case study.

Other tools that have been or are being created using BALBOA include one that analyzes the time between event occurrences, and one that incorporates an assertion processor for reasoning over event streams.

In addition, BALBOA supports many types of analyses that one might not normally think of as event-based. Since event attributes capture the resources involved in and the outcomes of the process actions, key process measurements that do not need event-level behavior information can still be calculated from the event data itself. For example, processing all "end-test-execution" events and tallying the outcome of each test can give a measurement of the success rate of testing. Such a metric is often used in deciding when to stop testing or to switch testing methods, and in predicting when software will be ready to release. Thus, many typical measures defined in existing process improvement paradigms (e.g., PSP [20]) could be supported by Balboa.

A number of possible enhancements to Balboa can be explored as future work.

- While event data supports many varieties of analyses, certainly there are other forms of valid data to be used. Although Balboa currently supports basic process attributes, a major direction to take Balboa in the future would be to incorporate other types of process, and perhaps product, data.
- A more extensive tool authorization mechanism would be useful, where a server has knowledge of who is allowed to connect to it, and from where. This information could be kept at a per collection level, allowing certain collections to be made public, while keeping others private.
- In the data collection interface, it would seem probable that events may not be reported at the exact time that they occur. Thus, building into Balboa some knowledge about the timestamps of events would allow it to ensure that an event stream is properly ordered, and would better support time-oriented queries on the client tool end.
- Hierarchical event collections would be useful when, for example, separate subprocesses are collected into separate collections, but there is a need to view the whole process as well.
- Sets of event collections, closely related to hierarchical collections, would be useful when analyzing many executions from a single process.
- Server-side filtering and processing of events could extend the capabilities of Balboa beyond simply mapping events and attributes. An embedded scripting language such as Tcl [24] would be ideal in providing the ability for tools to download scripts that manipulate and/or filter the event data before they are transmitted.
- A Web interface capability would enhance the interactivity and manageability of the server and data.

REFERENCES

[1] G.S. Avrunin, U.A. Buy, J.C. Corbett, L.K. Dillon, and J.C. Wileden. Automated Analysis of Concurrent Systems with the Constrained Expression Toolset. *IEEE Transactions on Software Engineering*, 17(11):1204–1222, November 1991.

[2] S. Bandinelli, A. Fuggetta, C. Ghezzi, and L. Lavazza. SPADE: An Environment for Software Process Analysis, Design, and Enactment. In A. Finkelstein, J. Kramer, and B. Nuseibeh, editors, *Software Process Modeling and Technology*, pages 223–248. Wiley, 1994.

[3] N.S. Barghouti and B. Krishnamurthy. Using Event Contexts and Matching Constraints to Monitor Software Processes. In *Proceedings of the 17th International Conference on Software Engineering*, pages 83–92. IEEE Computer Society Press, April 1995.

[4] V.R. Basili and H.D. Rombach. The TAME Project: Towards Improvement-oriented Software Environments. *IEEE Transactions on Software Engineering*, SE-14(6):758–773, June 1988.

[5] V.R. Basili and D.M. Weiss. A Methodology for Collecting Valid Software Engineering Data. *IEEE Transactions on Software Engineering*, 10(6):728–737, 1984.

[6] P. Bates. Debugging Heterogenous Systems Using Event-Based Models of Behavior. In *Proceedings of a Workshop on Parallel and Distributed Debugging*, pages 11–22. ACM Press, January 1989.

[7] I.S. Ben-Shaul and G.E. Kaiser. A Paradigm for Decentralized Process Modeling and it Realization in the Oz Environment. In *Proceedings of the 16th International Conference on Software Engineering*, pages 179–188. IEEE Computer Society Press, May 1994.

[8] G.A. Bolcer and R.N. Taylor. Endeavors: A Process System Integration Infrastructure. In *Proceedings of the Fourth International Conference on the Software Process*, pages 76–85. IEEE Computer Society, December 1996.

[9] M.G. Bradac, D.E. Perry, and L.G. Votta. Prototyping a Process Monitoring Experiment. *IEEE Transactions on Software Engineering*, pages 774–784, October 1994.

[10] J.E. Cook, L.G. Votta, and A.L. Wolf. Cost-Effective Analysis of In-Place Software Processes. *IEEE Transactions on Software Engineering*. To appear.

[11] J.E. Cook and A.L. Wolf. Toward Metrics for Process Validation. In *Proceedings of the Third International Conference on the Software Process*, pages 33–44. IEEE Computer Society, October 1994.

[12] J.E. Cook and A.L. Wolf. Automating Process Discovery through Event-Data Analysis. In *Proceedings of the 17th International Conference on Software Engineering*, pages 73–82. Association for Computer Machinery, April 1995.

[13] J.E. Cook and A.L. Wolf. Software Process Validation: Quantitatively Measuring the Correspondence of a Process to a Model Using Event-Based Data. Technical Report CU-CS-820-96, Department of Computer Science, University of Colorado, November 1996.

[14] J.E. Cook and A.L. Wolf. Discovering Models of Software Processes from Event-Based Data. *ACM Transactions on Software Engineering and Methodology*, 7(3), July 1998. To appear.

[15] J. Cuny, G. Forman, A. Hough, J. Kundu, C. Lin, L. Snyder, and D. Stemple. The Adriane Debugger: Scalable Application of Event-Based Abstraction. In *Proceedings of the ACM/ONR Workshop on Parallel and Distributed Debugging*, pages 85–95. ACM Press, May 1993.

[16] C. Gerety. HP SoftBench: A New Generation of Software Development Tools. Technical Report SESD–89–25, Hewlett-Packard Software Engineering Systems Division, Fort Collins, Colorado, November 1989.

[17] J. Grudin. Groupware and Cooperative Work: Problems and Prospects. In R.M. Baeker, editor, *Groupware and Computer-Supported Cooperative Work*, pages 97–105. Morgan Kaufmann, 1993.

[18] D. Heimbigner. The ProcessWall: A Process State Server Approach to Process Programming. In *SIGSOFT '92: Proceedings of the Fifth Symposium on Software Development Environments*, pages 159–168. ACM SIGSOFT, December 1992.

[19] D.M. Hilbert and D.F. Redmiles. An Approach to Large-Scale Collection of Application Usage Data Over the Internet. In *Proceedings of the 1998 International Conference on Software Engineering*. IEEE Computer Society Press, April 1998. (to appear).

[20] W.S. Humphrey. *A Discipline for Software Engineering*. SEI Series in Software Engineering. Addison-Wesley, 1995.

[21] B. Krishnamurthy and D.S. Rosenblum. Yeast: A General Purpose Event-Action System. *IEEE Transactions on Software Engineering*, 21(10):845–857, October 1995.

[22] R.J. LeBlanc and A.D. Robbins. Event-Driven Monitoring of Distributed Programs. *IEEE Transactions on Software Engineering*, 10(6):515–522, 1985.

[23] C.M. Lott. Process and Measurement Support in SEEs. *SIGSOFT Software Engineering Notes*, 18(4):83–93, October 1993.

[24] J.K. Ousterhout. *Tcl and the Tk Toolkit*. Professional Computing Series. Addison-Wesley, Reading, MA, 1994.

[25] S.P. Reiss. Connecting Tools Using Message Passing in the Field Environment. *IEEE Software*, pages 57–66, July 1990.

[26] R.L. Schwartz, P.M. Melliar-Smith, and F.H. Vogt. An Interval Logic for Higher-Level Reasoning. In *Proceedings of the Second ACM Symposium on Principles of Distributed Computing*, pages 173–186. ACM Press, August 1983.

[27] R.W. Selby, A.A. Porter, D.C. Schmidt, and J. Berney. Metric-Driven Analysis and Feedback Systems for Enabling Empirically Guided Software Development. In *Proceedings of the 13th International Conference on Software Engineering*, pages 288–298. IEEE Computer Society, May 1991.

[28] *An Introduction to the ToolTalk Service*. Sun Microsystems, Inc., 1991.

[29] A.L. Wolf and D.S. Rosenblum. A Study in Software Process Data Capture and Analysis. In *Proceedings of the Second International Conference on the Software Process*, pages 115–124. IEEE Computer Society, February 1993.

[30] P.H. Worley. A New PICL Trace File Format. Technical Report ORNL/TM-12125, Oak Ridge National Laboratory, 1992.