Context-based Analysis of System Execution Traces for Validating Distributed Real-time and Embedded System Quality-of-Service Properties

James H. Hill
Dept. of Computer and Information Science
Indiana University-Purdue University Indianapolis
Indianapolis, IN USA
Email: hillj@cs.iupui.edu

Abstract—System execution traces are useful artifacts for capturing distributed system behavior and validating quality-of-service (QoS) properties, such as end-to-end response time, throughput, and scalability. Although system execution traces assist in validating QoS properties, system execution traces are *very* dense. This is because system execution traces capture a system's complete behavior and execution in its target environment, which can consist of many components deployed across many hosts that execute for long periods of time. It therefore can be hard for distributed system testers to effectively analyze different views of QoS properties using system execution traces and/or pinpoint performance bottlenecks that need to be resolved.

This paper provides two contributions on validating distributed system QoS properties. First, this paper presents *Aspects for System Execution Traces (AsSET)*, which is a join point model for applying aspects to system execution traces and enabling context-based analysis of system execution traces. Secondly, it shows how applying AsSET to a representative enterprise distributed system can improve QoS analytical capabilities for distributed system testers. The tools and techniques discussed in this paper have been realized in an open-source system execution modeling tool named *CUTS*, which is currently used on several industry-related projects.

Keywords-context-based analysis, aspects, join point model, QoS validation, system execution traces

I. INTRODUCTION

Current trends and challenges. Enterprise distributed real-time and embedded (DRE) systems, such as shipboard computing environments, large-scale traffic management systems, and global information systems, are steadily increasing in size (*e.g.*, number of lines of source code and hardware/software resources) and complexity (*e.g.*, envisioned operational scenarios and target execution environment) [12], [29]. Because of this steady increase in both size and complexity, it is becoming more critical to validate their quality-of-service (QoS) properties (*e.g.*, end-to-end response time, scalability, and reliability) on the target architecture during early stages of the software lifecycle and continuously throughout it. Failure to identify performance bottlenecks early in the software lifecycle can result in elongated software lifecycles at the expense of overrun project budgets [1], [17], [26].

System execution modeling (SEM) [25] is one technique that enables DRE system testers to validate enterprise DRE system QoS properties in their target environment during early phases of the software lifecycle, *i.e.*, before system integration time. SEM tools achieve this goal by emulating realistic representations of the system under development in its target environment. During the emulation process, SEM tools capture system behavior using system execution traces [2], [14], [18], which are a collection of messages that capture the system's state at different times of execution. DRE system testers can then analyze the system execution traces and validate DRE system QoS properties, such as validating if a critical path of execution meets it specified end-to-end response time or the latency of an event is under its allowable threshold value.

Although system execution traces are an architecture-, language-, and technology-independent mechanism for validating DRE system QoS properties, it can be *hard* for DRE system testers to effectively exact meaningful information from them. For example, system execution traces can consist of hundreds of thousands of messages. Visually analyzing all the data from a system execution trace may not provide as meaningful information as viewing only a subset of the actual data. This is because critical pieces of information that can assist in analysis (*e.g.*, the context of a given data point) can be masked behind the noise of viewing data extracted from a system execution trace in totality (*e.g.*, visually analyzing a graph that is distorted because of outliers in the complete set of data). DRE system testers therefore need techniques that will improve their analytical capabilities of system execution traces when validating QoS properties.

Solution approach → Context-based analysis using aspects. *Context-based analysis* [5] is the process of analyzing a subset of a given dataset that meets a specified criteria, such as data points that fall within a specified range, or data points produced by a specified entity/component. It can also provide different viewpoints of data, which can provide different understandings problem domain. For example, in aspect-oriented programming (AOP) [15], context-based analysis is realized using aspects because aspects facilitate analysis of a concern by isolating it from core functionality. This can improve understandability of both core functionality and the concern captured as an aspect. In the context of validating DRE system QoS properties, context-based analysis can be

used to improve understanability when analyzing dense system execution traces by providing multiple views of the data.

This paper presents a methodology called *Aspects for System Execution Traces (AsSET)* that enables context-based analysis of system execution traces using AOP techniques. DRE system testers use AsSET by first defining an aspect that will isolate points-of-interest (or an apsect) in system execution traces from other data points. AsSET then data mines the system execution trace and extracts data related to the context defined by DRE system testers. Results from applying AsSET to a representative DRE system shows that AsSET helps reduce the amount of data DRE system testers have to analyze. Moreover, AsSET helps improve DRE system testers analytical capabilities because they can focus on isolated points-of-interest in the data without being distracted by noise associated with viewing data from system execution traces in totality.

Paper organization. The remainder of this paper is organized as follows: Section II introduces a case study of a representative DRE system; Section III describes the structure and functionality of AsSET; Section IV presents the results of applying AsSET to the case study; Section V compares AsSET with related work; and Section VI presents concluding remarks and lessons learned.

II. CASE STUDY: THE SLICE SCENARIO

The SLICE scenario is a representative DRE system from the domain of shipboard computing environments. The SLICE scenario has been used in prior case studies and research efforts, such as validating SEM tools, highlighting challenges of searching the deployment solution space for component-based systems, and developing architecture-independent approach for emulating CPU workload.
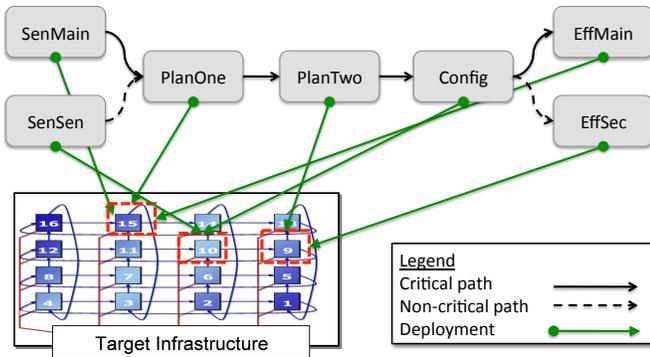

Fig. 1. Conceptual model of the SLICE scenario [9].

To briefly reiterate the SLICE scenario, Figure 1 presents an high-level overview of the SLICE scenario. As illustrated in this figure, the SLICE scenario consists of seven different components (*i.e.*, the rectangle figures): SenMain, SenSec, PlanOne, PlanTwo, ConfigOp, EffMain, and EffSec. Each of the seven components must be deployed on one of three hosts in the target environment highlighted by the dashed boxes on the Target Infrastructure in Figure 1. Finally, there is a

critical path of execution (*i.e.*, the solid lines between the components) with a end-to-end response time that must meet specified deadlines depending on the SLICE scenario's role in the case study. For this case study, however, there is no predefined end-to-end response time for the SLICE scenario's critical path of execution.

To assist with validating the SLICE scenario's critical path of execution, DRE system testers are using system execution modeling tools. In particular, they are using CUTS [10] to collect metrics via system execution traces, and then using UNITE [11] to data mine the system execution traces and generate QoS performance graphs to validate QoS properties (see Sidebar 1 for more details). The generated QoS performance graphs show DRE system testers the data trend for the SLICE scenario's critical path of execution over its complete lifetime, *i.e.*, how its performance changes with respect to time.

Sidebar 1: Overview of CUTS and UNITE

CUTS [10] is a system execution modeling tool that enables DRE system testers to conduct system integration test and validate QoS properties on the target architecture during early phases of the software lifecycle. DRE system testers use CUTS via the following steps:

1) Use domain-specific modeling languages [16] to model behavior and workload at high-levels of abstraction;
2) Use generative programming techniques [4] to synthesize a complete test system from constructed models that conform to the target architecture; and
3) Use emulation techniques to execute the synthesized system on its target architecture and validate its QoS properties, such as end-to-end response time, latency, and scalability, in its target execution environment.

DRE system testers can also replace emulated portions of the system with its real counterpart as its development is completed. This therefore allows DRE system testers to perform *continuous system integration testing*, *i.e.*, the process of execution system integration test to validate QoS properties continuously throughout the software lifecycle. CUTS supports validating QoS properties on several network communication architectures, such as: CIAO [13], OpenSplice [20], RTI-DDS [21], and TCP/IP.

UNITE [11] is a tool distributed with CUTS that enables DRE system testers to generate QoS performance graphs from system execution traces and validate QoS properties. DRE system testers use UNITE by first generating a system execution trace from emulating the system in its target environment. They then define a dataflow model [7] that is used to data mine metrics of interest for a given QoS property from the system execution trace, such as throughput of an event. Finally, UNITE constructs a QoS performance graph that shows that QoS property's data trend throughout the lifetime of the system (*i.e.*, how that QoS property changed with respect to time). Section III-A presents a more detailed overview of UNITE's methodology.

Although CUTS and UNITE enable DRE system testers to collect and extract metrics for validating the SLICE scenario's critical path of execution, it is hard for them to get meaningful information from the generated QoS performance graphs. More specifically, DRE system testers are faced with the

following challenges:

- **Challenge 1: Inability to extract only points-of-interest.** QoS properties change, *i.e.*, have different values, throughout an enterprise DRE system's execution lifetime. All the different values of a QoS property, however, will not be of interest a DRE system tester. For example, a DRE system tester may want to focus on data related to when an event sent between multiple components (as in the SLICE scenario) that did not meet its end-to-end response time.

  DRE system testers therefore need a technique that will allow them to identify and isolate points-of-interest (or a context) within system execution traces when generating QoS performance graphs. The generated QoS performance graphs should focus more on the specified context so DRE system testers are not distracted by other data captured in a system execution trace. Section III-B discusses how AsSET addresses this challenge by defining a join point model [15] for system execution traces.

- **Challenge 2: Inability to associate contexts with extracted points-of-interest.** Viewing identified points-of-interest in isolation does not provide DRE system testers with much meaningful information. This is because in order to analyze a point-of-interest, it is necessary to understand how it reached that point (*i.e.*, the data trend surrounding to the points-of-interest).

  DRE system testers therefore need a technique that will allow them to associate contexts with identified points-of-interest. This will help DRE system testers get more meaningful information from identified points-of-interest. Section III-C discusses how AsSET addresses this challenge using viewpoints, which allows DRE system testers to specify what data surrounding the isolated point-of-interest to include in the generated QoS performance graph.

Due to these challenges, it is hard for DRE system testers to use CUTS and UNITE to extract meaningful information from QoS performance graphs of the SLICE scenario's critical path of execution. Moreover, these challenges extend beyond the SLICE scenario, CUTS, and UNITE, and apply to other DRE systems that use system execution traces to validate QoS properties. The remainder of this paper therefore shows how AsSET addresses the challenges presented above and improves analytical capabilities for validating enterprise DRE system QoS properties using system execution traces.

III. AsSET: Applying Aspects to System Execution Traces

This section presents the structure and functionality of AsSET, which uses AOP techniques to facilitate context-based analysis of system execution traces when validating enterprise DRE system QoS properties. In addition, examples for the case study introduced in Section II are used to illustrate how AsSET can be applied to a representative DRE system. Before discussing the details of AsSET, however, it is necessary to provide an overview of how DRE system testers can validate

QoS properties using system execution traces to provide this section with some essential context.

A. An Overview of Validating QoS Properties via System Execution Traces

System execution traces are a collection of messages that capture a DRE system's behavior and state at any given point in time while executing in its target environment. Prior research efforts [11] yielded a tool called UNITE (see Sidebar 1) that data mines system execution traces to validate QoS properties, such as end-to-end response time, latency, and scalability. UNITE also has the ability to generate QoS performance graphs for QoS properties by viewing its data trend (*i.e.*, how a metric changes with respect to time).
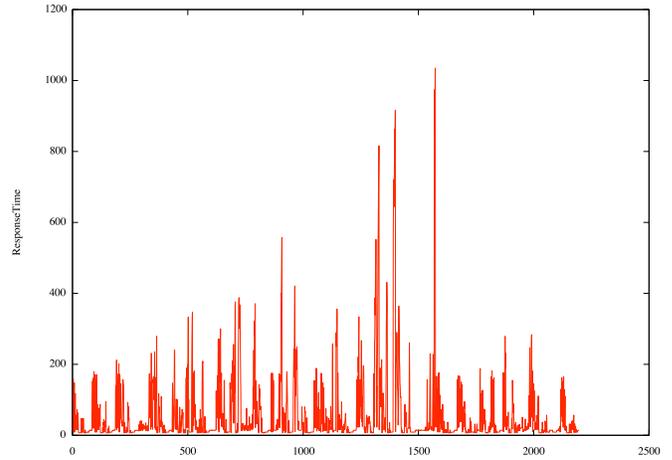

Fig. 2. Example QoS performance graph generated by UNITE.

Figure 2 shows a QoS performance graph for the SLICE scenario's critical path of execution. This graph was generated by viewing the data trend for metrics extracted from a system execution traces that represent the critical path's end-to-end response time. The benefit of viewing a data trend is it provides more meaningful information than a single scalar value that represents a QoS property, *e.g.*, calculating that the average end-to-end response time of the SLICE scenario's critical path of execution is 123.45 msec. Moreover, the data trend provides context to DRE system testers as to why the QoS property had the calculated result.

In order to validate a QoS property and view its data trend using system execution traces, UNITE uses *dataflow models* [7]. A dataflow model captures how data is passed (or flows) through a given domain—similar to how connections between separate components in a DRE system capture how data (or events) flows through a distributed system. In context of validating QoS properties using system execution traces, UNITE defines a dataflow model $DM = (LF, CR)$ as:

- A set LF of log formats that have a set V of variables identifying what data to extract from log messages in a system execution traces. These log formats will identify many occurrences of the same log message in a system execution trace where their difference is captured in V.

ID	Time of Day	Hostname	Severity	Message
45	2009-03-06 05:15:55	node1.isislab.vanderbilt.edu	INFO	Config: sent event at 5 at 120394455
46	2009-03-06 05:15:55	node1.isislab.vanderbilt.edu	INFO	Planner: sent event at 6 at 120394465
47	2009-03-06 05:15:55	node2.isislab.vanderbilt.edu	INFO	Planner: received event at 5 at 120394476
48	2009-03-06 05:15:55	node1.isislab.vanderbilt.edu	INFO	Config: sent event at 7 at 120394480
49	2009-03-06 05:15:55	node2.isislab.vanderbilt.edu	INFO	Effector: received event at 6 at 120394488
50	2009-03-06 05:15:55	node2.isislab.vanderbilt.edu	INFO	Planner: received event at 7 at 120394502

- A set CR of causal relations that specify the order of occurrence for each log format such that $CR_{i,j}$ means $LF_i \rightarrow LF_j$, or LF_i occurs before LF_j [24]. These relations help determine how data flows across different application domains, such as one component sending event to another component deployed on a different host.
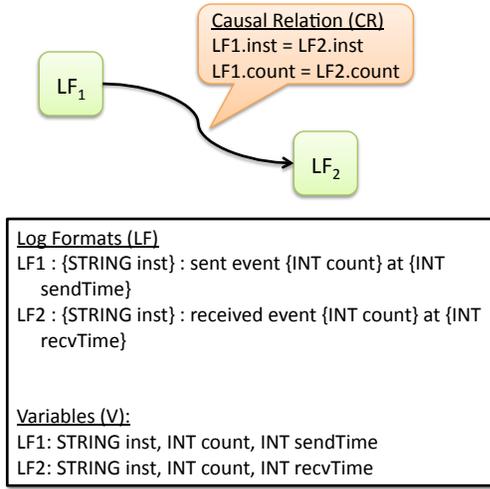

Fig. 3. Example dataflow model in UNITE.

Figure 3 shows the dataflow model, which is specified by DRE system testers, used to produce QoS performance graphs from system execution traces, such as the one illustrated in Table I. As shown in this figure, the two log formats will locate messages for the publication (*i.e.*, LF1 will identify message 45, 46, and 48 in Table I) and receipt (*i.e.*, LF2 will identify message 47, 49, and 50 in Table I) of an event. The variables in the two log formats capture the variable portion of each log message. These variables also contain metrics of interest.

$$LF2.recvTime - LF1.sendTime$$

Listing 1. Expression for calculating latency using UNITE.

Finally, the listing above presents the expression specified by DRE system testers to calculate the latency for sending an event based on the data captured in a system execution trace. UNITE then uses the dataflow model and system execution trace to evaluate the user-defined QoS validation equation, and optionally view the data trend of the QoS property as shown in Figure 2.

If a DRE system tester wants to view a subset of the data (*i.e*, the portion of the graph in Figure 4 enclosed in the dashed
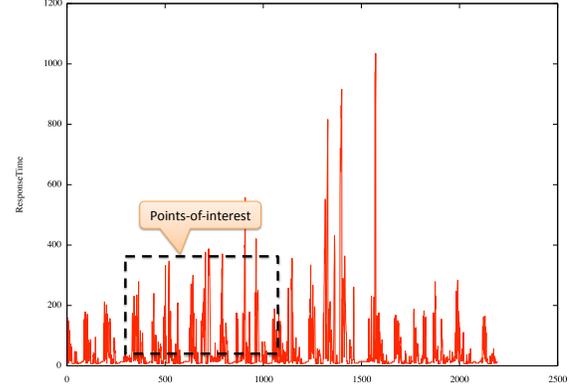

Fig. 4. Example data trend with points-of-interest highlighted.

rectangle), then they have to use *ad hoc* techniques, such as manually augmenting the data to graph only a subset of it. Although this is feasible, it can be a tedious, time-cosuming, and error-prone process. Moreover, as the system execution trace increases in size (*i.e*, the size of the dataset) and complexity (*i.e.*, the process associated with analyzing the system execution trace), it become harder to use *ad hoc* techniques to extract meaningful information from system execution traces. The remainder of this section therefore discusses how AsSET and its context-based analysis methodology improve analytical capabilities of system execution traces.

B. Defining a Join Point Model for System Execution Traces

The essence of AOP is to capture and separate cross-cutting concerns, such as the logging context of an application, from core functionality of the software system. For example, Listing 2 shows an example that uses AspectJ to capture cross-cutting concerns for logging.

```
// AspectJ code
before() :
  execution(void HelloWorld.sayGreeting()) {
  System.out.println (
    ''Entering sayGreeting()'');
}

after() :
  execution(void HelloWorld.sayGreeting()) {
  System.out.println(
    ''Exiting sayGreeting()'');
}

// target Java code
public class HelloWorld {
```

```
    ...

    void sayGreeting () {
        // do something ...
    }
}
```

Listing 2. Example AspectJ code for capturing the logging cross-cutting concern in software systems.

As highlighted in this example, logging operations for tracing entry/exit into/from the `HelloWorld.sayGreeting()` method is captured as an aspect. AspectJ then *weaves* the aspect into the target code bases to produce the result shown in Listing 3. As illustrated in this listing, the core functionality of the software system now contains code related to the logging aspect.

```
// final Java code
public class HelloWorld {
    ...

    void sayGreeting () {
        System . out . println (
            ''Entering sayGreeting ()'');

        // do something

        System . out . println (
            ''Exiting sayGreeting ()'');

    }
}
```

Listing 3. Example of weaving AspectJ code into an existing code base.

Irrespective of what AOP technology used (*e.g*, AspectJ [15] or AspectC++ [27]), the underlying fundamentals of each technique is centered around a *join point model* [15]. The join point model is an advice-based methodology that allows system developers to define:

- **pointcuts** – pointcuts determine point of execution in the application at which cross-cutting concern needs to be applied, *e.g.*, `after()` AspectJ statement in Listing 2; and
- **advice** – advice specifies what code should run at the pointcut, such as the `System.out.println(...)` methods captured in each aspect in Listing 2.

In order to apply aspects to system execution traces and enable context-based analysis of QoS properties, it is necessary to first define a join point model for this domain. In the context of system execution traces—and building upon the definition of a dataflow model DM presented in Section III-A—the join point model is defined as:

- **pointcuts** – pointcuts are the values of variables in the dataflow model DM that determine when to apply the specified aspect, such as when a variable or an expression using variables in dataflow model have a specific value; and
- **advice** – advice determines how to execute the pointcut. For context-based analysis, the advice will always remove data points from the complete set of data.

Realizing the join point model in AsSET. To realize aspects in AsSET using the definition above, DRE system testers first select variables from the dataflow model used to data mine a system execution trace. For example, using the dataflow model in Figure 3, DRE system developers can use `LF1.sendTime` and `LF2.recvTime` to isolate points-of-interest related to latency.

After selecting what variables to use from the dataflow model, DRE system testers define an boolean expression using the identified variables that determine when the pointcut is to execute. Listing 4 shows an example pointcut for selecting metrics for when the latency is less than 3 time units (or msec).

```
LF2 . recvTime − LF1 . sendTime < 3
```

Listing 4. Example pointcut for isolating points-of-interest in system execution traces.

Likewise, Listing 5 illustrates another pointcut for selecting points-of-interest related to when the latency is less than 3 time units and the event count is greater than 40. As illustrated in Listing 4 and Listing 5, DRE system developers have the flexible to define the pointcut in terms of any state that can be identified in the system execution trace.

```
LF2 . recvTime − LF1 . sendTime < 3 AND
LF2 . count > 40
```

Listing 5. Example of a complex boolean expression join point model for isolating points-of-interest in system execution traces.

The final step is specifying how to run the code, or the *advice*. AsSET accomplishes this by converting the pointcut to an SQL expression. This is done by creating a standard `SELECT` statement where the pointcut is the boolean expression in the `WHERE` clause. For example, Listing 6 illustrates the SQL representation for the aspect in Listing 4.

```
SELECT ∗ FROM dataset
    WHERE LF2 . recvTime − LF1 . sendTime < 3
```

Listing 6. SQL statement representation of an aspect in AsSET.

The resultant SQL expression is then applied to the original dataset constructed from data mining the system execution trace using UNITE. The end result of this process is isolated points-of-interest (if any) from the system execution trace that matches the aspect defined by DRE system testers, thereby addressing Challenge 1 introduced in Section II.

C. Defining Context for Extracted Points-of-Interest

Section III-B defined a join point model for isolating points-of-interest in system execution traces to improve QoS analytical capabilities. Although the join point model is able to reduce the amount of data that DRE system must analyze (*i.e.*, enables them to focus on a subset of the complete data), it is hard to DRE system testers to get meaningful information from the isolated points (except for the fact that the points-of-interest exist in the data). This is because there is no context associated with each individual point-of-interest to help DRE system testers understand its true meaning (*i.e.*, how it evolved to that point).

To provide more meaningful information to DRE system testers, it is therefore necessary to extend the definition of the

join point model discussed in Section III-B with a methodology that allows DRE system testers to associate context with isolated points-of-interest. In context of applying aspects to system execution traces, the concept of a *viewpoint* is used to give meaningful context to isolated points-of-interest that are result of applying an aspect to system execution traces. In particular, a viewpoint $VP = (n, m)$ is defined as a 2-tuple such that if viewing the complete data trend, then n is the number of data points before and m is the number of data points after each isolated point-of-interest that will provide meaningful context.
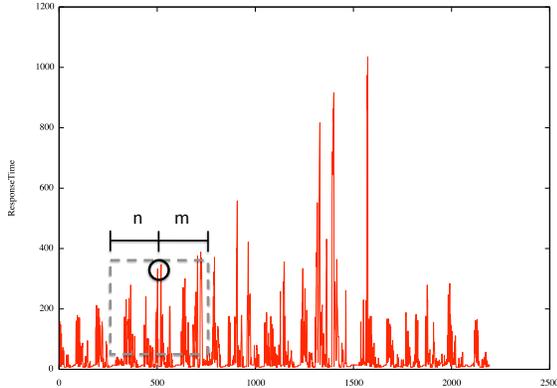

Fig. 5. Example illustrating the concept of a viewpoint.

Figure 5 illustrates the concept of a viewpoint. As shown in this figure, the circle on the data trend represents the point-of-interest that will be isolated after applying the aspect to the system execution trace. Likewise, the dashed box embedded in the figure represents the viewpoint where the left side of the box is n data points before the isolated point-of-interest and right side of the box is m points after the isolated point-of-interest. Figure 6 shows the final result of applying the aspect and viewpoint to the system execution trace, which is easier for DRE system testers to comprehend when compared the same context appearing in Figure 5.
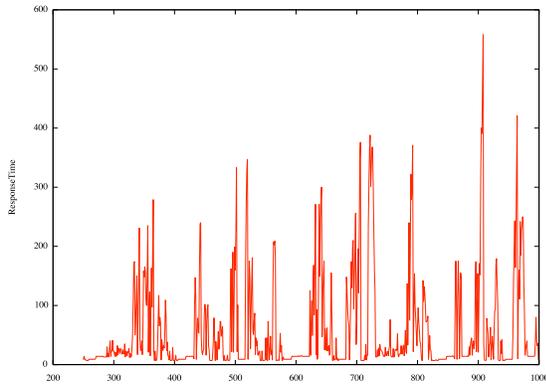

Fig. 6. Result of applying a viewpoint to a join point model.

The different types of view. Applying an aspect to system execution traces can result in many isolated points-of-interest.

Likewise, the size of the viewpoint, which is user-defined, can result in many different ways to add context to a given point-of-interest. In particular, applying aspects and viewpoints to system execution traces to can result in the following views of isolated points-of-interest:

- **Null view.** This view occurs when an aspect is applied and results in no isolated points-of-interest. For example, specifying a join point model that will isolate points-of-interests where the latency of an event is less than 30 msec, but analysis of the system execution trace shows that never occurred during the system's execution.
- **Scalar view.** This view occurs when an aspect is applied to a system execution trace and results in a single point-of-interest. Likewise, applying a viewpoint will result in a single data trend—similar to that shown in Figure 6.
- **Piecemeal view.** This view occurs when an aspect is applied to a system execution trace and results in more than one isolated point-of-interest. Likewise, applying a viewpoint to the isolated points-of-interest results in a piecemeal graph of the overall data trend. Figure 7a illustrates an example of a piecemeal view that results from applying aspects and viewpoints to a system execution trace.
- **Overlapping view.** This view occurs when application of an aspect results in more than one isolated point-of-interest, similar to the piecemeal view discussed above. The main difference with the overlapping view is that the viewpoint results in views that overlap with each other as shown in Figure 7b. It is also possible to have both overlapping and piecemeal views occur simultaneously after applying an aspect and viewpoint to a system execution trace.
- **Complete view.** This view occurs when an aspect is applied to a system execution trace and results in one or more points-of-interest. Likewise, applying a viewpoint to points-of-interest result in viewing all the data, as opposed to a subset of the actual data. The complete view is equivalent to viewing a data trend without applying an aspect and viewpoint.

Distinguishing between scope and state. Using the current definition of a join point (including the viewpoint) to define aspects for context-based analysis of system execution traces, there can be cases when unwanted data is extracted from the system execution trace. For example, consider the aspect defined in Listing 4. In this example, points-of-interest where latency is less than 3 time units are extracted.

```
LF2.recvTime − LF1.sendTime < 3 AND
  LF2.inst = ``Config''

viewpoint: (5, 10)
```

Listing 7. Example aspect that confines aspect to a specific entity.

Now, consider the same example, but this time with an additional condition that confines points-of-interest to a specific entity. As shown in Listing 7, the aspect should only extract points-of-interest where the LF2.inst = ``Config''.
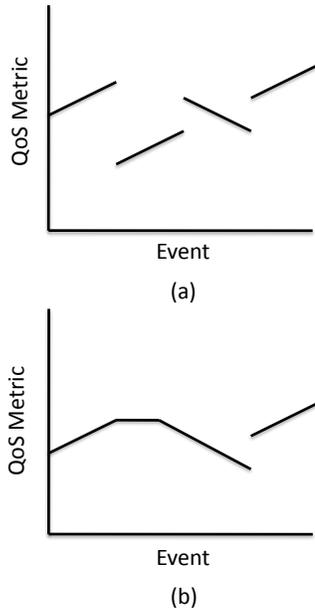
Fig. 7. Example of a (a) piecemeal view and (b) overlapping view.

Likewise, the viewpoint should extract 5 data points before and 10 data points after each identified point-of-interest.

Because data points for different entities can be intertwined in a system execution trace (as shown in Table I), there is great chance that the current approach for applying aspects to system execution traces will result in extracting invalid data (*e.g.*, data points where `NOT(LF2.inst = ''Config'')`). This is because the viewpoint by itself only looks at points before and after identified points-of-interest. The viewpoint does not concern itself with the actual contents of the data points, such as the value of `LF2.inst`.

To handle this challenge of ensuring only valid data is selected for a given aspect, the aspect's pointcut is broken down into the following two elements:

- **Scope** – The scope defines who does the aspect apply to (*i.e.*, what entities in the system execution trace); and
- **State** – The state defines what property values should the aspect extract from the system execution trace.

```
scope: LF2.inst = ''Config''
state: LF2.recvTime − LF1.sendTime < 3
viewpoint: (5, 10)
```

Listing 8. Example aspect using redefinition of a pointcut.

Using the above redefinition of a pointcut, Listing 8 highlights the correct specification of the aspect from Listing 7. As shown in the listing, the scope is set to `LF2.inst = ''Config''` and the state is defined as `LF2.recvTime − LF1.sendTime < 3`. Now, when applying the aspect to the system execution trace, it will not extract data points within the viewpoint that are not of the correct scope. The correctness of this process, however, is the sole responsibility of DRE system testers to correctly identify what variables and/or expressions belong in the scope and state specification.

Realizing contexts in AsSET. To realize contexts in AsSET, DRE system testers first define the join point model as shown above using the redefinition of a pointcut. After specifying the join point model, they then define a viewpoint that specifies the number of data points before and after isolated points-of-interest to include for contextual purposes. As illustrated in Listing 8, the viewpoint will include 5 data points before and 10 data points after each isolated point-of-interest. Because a viewpoint may not contain a contiguous set of data points from the original data set due to intertwined data points from other scopes, the algorithm for applying aspects to system execution traces is not trivial.

Algorithm 1 General algorithm for applying aspects to system execution traces.

1: **procedure** APPLY_ASPECT(DS, A)
2:     DS: dataset constructed by UNITE
3:     A: aspect to apply to dataset
4:     CREATE TEMP TABLE _subset_ AS SELECT ROWID AS real_rowid, * FROM DS WHERE SCOPE(A) GROUP BY SCOPE_VARIABLES(A);
5:
6:     CREATE TEMP TABLE _ranges_ AS SELECT position - viewpoint_lower(A) AS lower, position + viewpoint_upper(A) AS upper FROM (SELECT ROWID AS position FROM _subset_ WHERE STATE(A));
7:
8:     SELECT * FROM _subset_ WHERE RANGES ('ROWID', '_ranges_', 'upper', 'lower');
9:     DROP TABLE _subset_;
10:     DROP TABLE _ranges_;
11: **end procedure**

Algorithm 1 presents the general algorithm for realizing aspects in AsSET as a sequence of SQL statements. As shown in this algorithm, given the original dataset DS constructed by UNITE, which is a simple SQL data table where each variable in the dataflow model is a column, and the aspect A, AsSET first selects the subset of the dataset that matches the specified scope and orders the data by the variables in the scope. This is necessary to segregate data points from different scopes, such as different values of `LF2.inst`. Using the subset of data, AsSET then selects data points that match the aspect's state, and assigns each data point (or point-of-interest) a unique id sequentially. This is necessary for selecting continuous data points within the viewpoint.

Using the unique ids assigned to the identified points-of-interest, AsSET constructs a table of ranges. The ranges in this table specify what data points in the subset are associated with a given viewpoint for a point-of-interest in the aspect. Finally, AsSET selects the appropriate data points from the subset using the identified ranges via the user-defined SQLite function $RANGES(x, table, lowerCol, upperCol)$:

- x – the column to apply ranges against, such as 'ROWID' in Algorithm 1;

- *table* – the name of the table that contains the ranges, such as `'_ranges_'` in Algorithm 1;
- *lowerCol* – the name of the column in *table* that contains the lower bound of the ranges, such as `'lower'` in Algorithm 1; and
- *upperCol* – the name of the column in *table* that contains the upper bound of the ranges, such as `'upper'` in Algorithm 1.

that constructs a disjunction of SQL `BETWEEN` statements. For example, if the range table contained the following values $\{(2, 6), (7, 20)\}$, then the resultant SQL clause would be `(x BETWEEN 2 AND 6) OR (x BETWEEN 7 AND 20)` where x is the name of the column to apply the `BETWEEN` statements against. By using the algorithm presented in Algorithm 1, AsSET is able accurately perform context-based analysis of system execution traces, thereby addressing Challenge 2 introduced in Section II.

IV. APPLYING AsSET TO THE SLICE SCENARIO

This section presents results from applying AsSET to the SLICE scenario introduced in Section II. This section also empirically quantifies the cost-effectiveness of using AsSET to apply aspects to system execution traces.

A. Experimental Setup

Section II introduced the SLICE scenario case study. As discussed in that section, the SLICE scenario consist of 7 components. Likewise, there is a critical path of execution that must be completed within a specified deadline (*i.e.*, have a end-to-end response time under a specified upper bound).

Although the SLICE scenario does not have a specified deadline for its critical path of execution, DRE system testers want the ability to locate deployments that will give them the best end-to-end response time for a given configuration. This means that DRE system testers must test many different deployment and configurations and identify ones that meet their needs. Moreover, this implies that DRE system testers need tools that will assist them with analyzing QoS properties, and getting meaningful information results. This is especially critical when the amount of data generated by the SLICE scenario is too much to analyze in its totality.

Because DRE system testers are using CUTS and UNITE to validate the end-to-end response time of the SLICE scenario's critical path of execution, they decide to leverage AsSET to assist with analyzing the results. In particular, they are using AsSET evaluate only a subset of the data in generated system execution traces. This will assist them in focusing their analysis on key points in the data, such as viewing what component(s) are the bottleneck and understanding why they are the bottleneck.

Using AsSET, DRE system testers first defined aspects using the technique discussed in Section III-B. DRE system testers then defined viewpoints (as discussed in Section III-C) to provide context to the isolated points-of-interest. For completeness, all tests were run in a representative testbed based on ISISlab (www.isislab.vanderbilt.edu), which is powered by

Emulab software [22][1]. Each host in the experiment was an IBM Blade Type L20, dual-CPU 2.8 GHz processor with 1 GB RAM configured with the Fedora Core 6 operating system. The remainder of the section discusses the results for applying AsSET to SLICE scenario and assisting in analyzing its critical path of execution.

B. Experimental Results

After running several experiments of the SLICE scenario and evaluating its critical path of execution, DRE system testers wanted to get a better understanding of the results. DRE system testers therefore first graphed the data trend for each component in the SLICE scenario's critical path of execution.
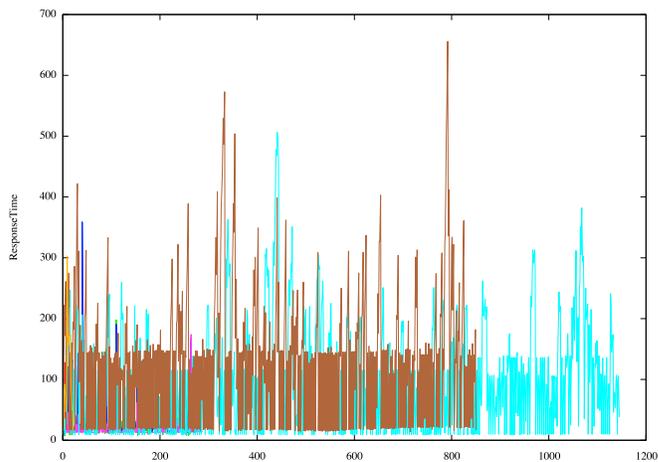

Fig. 8. Viewing the data trend of the response time for each component in the SLICE scenario.

As illustrated in Figure 8, using a single graph to display the data trend for each component is hard to analyze. For example, the maximum value of the y-axis is 700 msec because this will show the maximum value in the complete dataset. Likewise, the data trends are cluttered together, which makes it hard for DRE system testers to identify the data trend for each component.

Because Figure 8 does not provide meaningful information to DRE system testers who are evaluating the SLICE scenario's critical path of execution, they use AsSET to improve their analytical capabilities. In particular, DRE system testers wanted to learn if the data trend for each component will convey some meaningful information for analysis. DRE system testers therefore defined an aspect where the pointcut isolated points pertaining to a single component in the system, such as SenMain.

Figure 9 therefore highlights the a complete view (see Section III-C) for the SenMain component's response time in the SLICE scenario where the join point model selects data points for only this particular component. As illustrated in

---

[1]Emulab allows developers and testers to configure network topologies and operating systems on-the-fly to produce a realistic operating environment for distributed unit and integration testing.
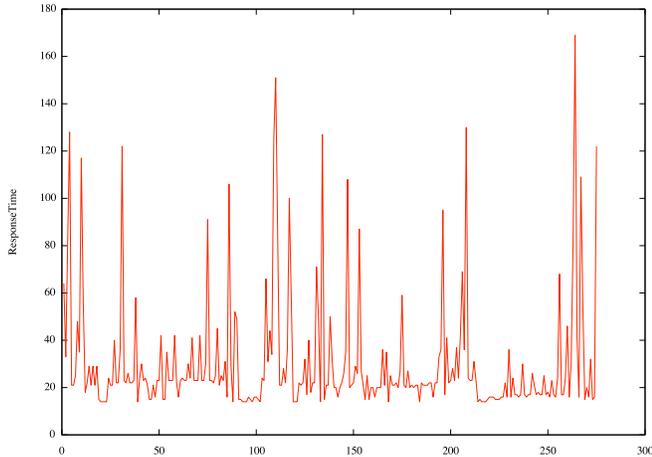
Fig. 9. Viewing the response time of a single component in the SLICE scenario.

this figure, it is easier to comprehend in comparison to the results presented in Figure 8. For example, the maximum y-axis value is 180 msec instead of 700 msec. Likewise, the graph in Figure 9 is not cluttered with other data (*i.e.*, there is less noise for DRE system testers to filter). DRE system testers therefore can get improved understanding of QoS performance graphs for analytical purposes using AsSET and its aspect-oriented programming techniques.

C. Quantitative Analysis of AsSET

Section IV-B presented results that showed AsSET can improve understanding of QoS performance graphs using aspects. This improved understanding is accomplished by reducing the amount of data DRE system testers must analyze, and allowing them to focus on isolated points-of-interest.

From a graphicial point-of-view, it is straightforward to see how AsSET provides an effective approach for assisting in the analysis of DRE system QoS properties. From a process point-of-view, however, it is necessary to quantify the cost-effectiveness of AsSET's methodology. Table II therefore presents the steps DRE systems must take when using *ad hoc* and *ad hoc* techniques versus using AsSET for isolating points-of-interest to improve understanding QoS performance graphs.

TABLE II
COMPARISON OF STEPS REQUIRED TO ISOLATE POINTS-OF-INTEREST IN
QoS PERFORMANCE GRAPHS USING *ad hoc* TECHNIQUES VERSUS USING
ASSET.

Step	Ad hoc Techniques
1	Create dataset
2	Manually locate points-of-interest
3	Manually locate context data for each point-of-interest
Step	**Aspects for System Execution Traces (AsSET)**
1	Create dataset
2	Define aspect and viewpoint
3	Apply aspect to dataset

As illustrated in Table II, the number of steps for isolating

points-of-interests in QoS performance graphs is conceptual the same for either approach. Although the number of steps in this table is the same, in practice the amount of physical time and effort using AsSET is less than *ad hoc* techniques. This is because Step 2 and 3 for *ad hoc* techniques is an iterative process that requires *think time* [25]. In AsSET, Step 3, which is the equivalent of Step 2 and 3 for *ad hoc* techniques is automated. Moreover, Step 3 for AsSET is doing using SQL queries, which presents opportunities for optimizing the algorithm presented in Section III-C using more complex SQL statements, user-defined functions, or stored procedures. It is therefore possible to conclude that using AsSET to improve understanding to QoS performance graphs is more cost-effective than using *ad hoc* techniques, such as manually selecting and manipulating the data as currently done.

V. RELATED WORKS

This section compares AsSET with other related work. In particular, AsSET is compared to other works on context-based analysis and application of aspects outside of traditional AOP.

Content-based analysis. Content-based analysis has been used in many different application domains. For example, content-based analysis has been used to detect the presence of spam [19], and in security to detect policy violations [28]. Likewise, it is used in complex event process to identify events of interest [6]. AsSET therefore adds to the application domain using context-based analysis to remove unwanted to data when analyzing QoS properties to improve a domain-specific functionality.

Application of apsects for multiple views. Model-driven engineering [23] tools such as the Generic Modeling Environment (GME) [16] and Motorola WEAVR [3] use aspects to to provide multiple view of the domain. For example, GME has a concept called "aspect" that shows model elements only for that concern to model developers. Likewise, WEAVR has a join point model for UML that enables model developers to hide/show elements for different concerns. AsSET is similar to both GME and WEAVR in that it uses aspects to reduce the amount of data presented to DRE system testers. The result of all approaches is increased understanding of the domain by presenting only necessary information to the end-user.

Computer graphics and visualization. The concept of a *viewpoint* is well-known in the domain of computer graphics and visualization [8]. The viewpoint in their domain represents a point-of-view for a camera in the world. Similar to computer graphics and visualization, AsSET uses viewpoints to define a DRE system tester's point-of-view of data. Unlike computer graphics and visualization, AsSET use aspects to realize viewpoints instead based on relational databases and sets as opposed to complex mathematical operations.

VI. CONCLUDING REMARKS

Validating enterprise DRE system QoS properties requires tools and techniques that will help improve DRE system testers analytical capabilities and understanding of the data. Lack of such support hinders DRE system testers ability to

identify key pieces of data that could assist with resolving performance bottlenecks in timely and cost-effective manner. This paper presented *Aspects for System Execution Traces (AsSET)*, which uses aspect-oriented programming techniques to improve DRE system tester's analytical capabilities by isolating points-of-interests in the complete dataset. Moreover, context is associated with the isolated points-of-interest to improve understandability. By using AsSET, DRE system testers can focus more on analyzing QoS properties, instead of dealing with complexities associated with manually identifying and manipulating large amounts of data.

The following is a list of lessons learned from developing and applying AsSET to a representative enterprise DRE system project:

- **Conventional aspect-oriented programming techniques do not suffice for system execution traces.** This is because the join point model only isolates points-of-interest. It is necessary, however, to provide context with the isolated points-of-interest to improve understandability of the QoS properties. Extending the join point model to use viewpoints therefore helped address this concern and provide meaningful context to the isolated points-of-interest.

- **Dynamic and multi-aspects are needed to further improve analytical capabilities.** This is because there can be times when context needed to provide meaningful information may vary for each point-of-interest. Likewise, DRE system testers may want to apply more than one aspect to a system execution trace. Future work will therefore investigate techniques for realizing dynamic and multi-aspects for system execution traces.

- **Aspect-oriented programming techniques can be used to remove content.** Although this goes against conventional methodology of aspect-oriented programming techniques, this notion can hold true in other domains, such as viewing QoS performance graphs constructed from data mining system execution traces.

CUTS, UNITE, and AsSET are freely available in open-source format for download from the following location: www.cs. iupui.edu/CUTS.

REFERENCES

[1] J. T. Catanio. *Requirements Analysis: A Review*, pages 411–418. Springer Netherlands, 2006.

[2] F. Chang and J. Ren. Validating system properties exhibited in execution traces. In *ASE '07: Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering*, pages 517–520, New York, NY, USA, 2007. ACM.

[3] T. Cottenier, A. van den Berg, and T. Elrad. Stateful Aspects: The Case for Aspect-Oriented Modeling. In *AOM '07: Proceedings of the 10th International Workshop on Aspect-Oriented Modeling*, pages 7–14, 2007.

[4] K. Czarnecki and U. W. Eisenecker. *Generative Programming: Methods, Tools, and Applications*. Addison-Wesley, Reading, Massachusetts, 2000.

[5] F. J. Damerau. Automatic Parsing for Content Analysis. *Communications of the ACM*, 13(6):356–360, 1970.

[6] P. Dekkers. Complex Event Processing. Master's thesis, Radboud University Nijmegen, Nijmegen, Netherlands, October 2007.

[7] E. Downs, P. Clare, and I. Coe. *Structured Systems Analysis and Design Method: Application and Context*. Prentice Hall International (UK) Ltd., Hertfordshire, UK, UK, 1988.

[8] D. D. Hearn and M. P. Baker. *Computer Graphics with OpenGL*. Prentice Hall, 3 edition, 2003.

[9] J. H. Hill. An Architecture Independent Approach to Emulating Computation Intensive Workload for Early Integration Testing of Enterprise DRE Systems. In *Proceedings of the 11th International Symposium on Distributed Objects, Middleware, and Applications*, Vilamoura, Algarve-Portugal, November 2009.

[10] J. H. Hill, J. Slaby, S. Baker, and D. C. Schmidt. Applying System Execution Modeling Tools to Evaluate Enterprise Distributed Real-time and Embedded System QoS. In *Proceedings of the 12th International Conference on Embedded and Real-Time Computing Systems and Applications*, Sydney, Australia, August 2006.

[11] J. H. Hill, H. A. Turner, J. R. Edmondson, and D. C. Schmidt. Unit Testing Non-functional Concerns of Component-based Distributed Systems. In *Proceedings of the 2nd International Conference on Software Testing, Verification, and Validation*, Denver, Colorado, Apr. 2009.

[12] S. E. Institute. Ultra-Large-Scale Systems: Software Challenge of the Future. Technical report, Carnegie Mellon University, Pittsburgh, PA, USA, June 2006.

[13] Institute for Software Integrated Systems. Component-Integrated ACE ORB (CIAO). www.dre.vanderbilt.edu/CIAO/, Vanderbilt University.

[14] N. Joukov, T. Wong, and E. Zadok. Accurate and Efficient Replaying of File System Traces. In *FAST'05: Proceedings of the 4th conference on USENIX Conference on File and Storage Technologies*, pages 25–25, 2005.

[15] I. Kiselev. *Aspect-Oriented Programming with AspectJ*. Sams, 2002.

[16] Á. Lédeczi, Á. Bakay, M. Maróti, P. Völgyesi, G. Nordstrom, J. Sprinkle, and G. Karsai. Composing Domain-Specific Design Environments. *Computer*, 34(11):44–51, 2001.

[17] J. Mann. *The Role of Project Escalation in Explaining Runaway Information Systems Development Projects: A Field Study*. PhD thesis, Georgia State University, Atlanta, GA, 1996.

[18] D. Narayanan. End-to-end Tracing Considered Essential. In *Proceedings of High Performance Transactions Systems*, September 2005.

[19] Y. Niu, Y.-M. Wang, H. Chen, M. Ma, and F. Hsu. A Quantitative Study of Forum Spamming Using Context-based Analysis. In *Proceedings of the 14th Annual Network and Distributed System Security Symposium*, pages 692–696, San Diego, CA, February 2007.

[20] Prism Technologies. OpenSplice Data Distribution Service. www.prismtechnologies.com/, 2006.

[21] Real-time Innovations. NDDS: The Real-time Publish-Subscribe Middleware. www.rti.com/products/ndds/ndwp0899.pdf, 1999.

[22] R. Ricci, C. Alfred, and J. Lepreau. A Solver for the Network Testbed Mapping Problem. *SIGCOMM Computer Communications Review*, 33(2):30–44, Apr. 2003.

[23] D. C. Schmidt. Model-Driven Engineering. *IEEE Computer*, 39(2):25–31, 2006.

[24] M. Singhal and N. G. Shivaratri. *Advanced Concepts in Operating Systems*. McGraw-Hill, Inc., New York, NY, USA, 1994.

[25] C. Smith and L. Williams. *Performance Solutions: A Practical Guide to Creating Responsive, Scalable Software*. Addison-Wesley Professional, Boston, MA, USA, September 2001.

[26] A. Snow and M. Keil. The Challenges of Accurate Project Status Reporting. In *Proceedings of the 34th Annual Hawaii International Conference on System Sciences*, Maui, Hawaii, 2001.

[27] O. Spinczyk and D. Lohmann. The Design and Implementation of AspectC++. *Knowledge-Based Systems*, 20(7):636–651, 2007.

[28] K. Wan, V. Alagar, and Z. Y. Yang. A Context-Based Analysis of Intrusion Detection for Policy Violation. In *Proceedings of the International Conference on Computational Intelligence and Security*, pages 692–696, Washington, DC, USA, 2007. IEEE Computer Society.

[29] N. Wang, D. C. Schmidt, A. Gokhale, C. Rodrigues, B. Natarajan, J. P. Loyall, R. E. Schantz, and C. D. Gill. QoS-enabled Middleware. In Q. Mahmoud, editor, *Middleware for Communications*, pages 131–162. Wiley and Sons, New York, 2004.