Connecting Enterprise Applications to Metric Driven Verification

Author: Matt Graham, Cadence Design Systems, Inc, Ottawa Canada mgraham@cadence.com

Organizer: John Brennan, Cadence Design Systems, Inc, Chelmsford MA USA
brennanj@cadence.com

Adviser: Gergely Sass, Verification Engineer, NXP Semiconductor Austria GmbH
gergely.sass@nxp.com

***Abstract*— The DV community has been defining, applying and extending the concept of Metric Driven Verification methodology for some time. Its application at the IP/sub-system level is commonplace, and its expansion into the SoC integration level is ongoing. This expansion is thanks in no small part to advancements in EDA tools to manage the considerable amounts of metric (coverage, regression, etc) data generated by EDA tools (simulation, emulation, formal, etc). While these tools can help with the metrics that are directly generated by EDA, they often lack any meaningful integration with other Enterprise Applications such as defect tracking, requirements management, and source code changes and revision control, to name just a few.**

As EDA enhances their MDV applications to take advantage of database structures, and multi-user collaboration based architectures, as many of the Enterprise Applications mentioned above already have, an opportunity exists for tighter integration between those management systems. If the example of defect tracking is considered, a number of integration flows may be of interest. At the first order level, straightforward flows such as opening a defect report based on a regression failure, or associating sets of regression failures with existing defect reports. Second order flows would include rerunning any regression failures upon resolution or a defect report. A third order of integration would entail marking that a version or release of a design and testbench combination is in "debug" mode. Such a marking would indicate to the regression flow to not exercise any regressions on that version. This paper will describe in detail these flows, and their associated technical implementation with examples using specific industry standard tools.

Beyond these relatively obvious integrations lies the opportunity for analytics that move beyond transactional type operations into more abstract data analysis. One example is the analysis of coverage information collected for a specific set of regression failures associated to a specific defect report. This coverage correlation could then be exploited to seed further regressions that focus on the specific coverage areas to attempt to flush out more defects in a specific solution space. This paper will explore these types of integrations and discuss the technical requirements and challenges for their implementation.

Keywords—Metric Driven Verification, Functional Verification, EDA, Enterprise Applications, REST, ALM

I. Introduction:

Verification engineers continue to manage ever increasing amounts of data, in the forms of regressions, coverage, requirements, defect and release information. Each of these silos of information is of particular use and interest to the verification team, and each are an integral part of the closure decision. To this point, however, none have been particularly well integrated into a single reporting mechanism. Each form of data has a number of options in terms of applications for storing and reporting, but the aggregation of such information remains, for the most part, a manual process.

II. Connecting Enterprise Data to Verification

The desire to connect data from enterprise applications such as defect/bug tracking, requirements management, and others is not new. Many verification teams have been importing data into the verification process or exporting data from the verification process for some time. To date, any data exchange has been primarily ad hoc, requiring a considerable amount of manual work and management on the users part. In the best case, exchange of data via CSV files or other non-optimal formats enables the data to be exchanged in a somewhat mechanical manner. In the worst case, engineers are forced to cut and paste or manually massage the data for exchange.

III. Next Generation Planning and Management Solutions

For over a decade, EDA vendors have been providing verification planning and management (VPM) solutions enabling semi conductor design teams to effectively and efficiently apply metric driven verification MDV to their verification efforts. Until recently, VPM solutions primarily relied on standard files systems (e.g. NFS) for storing any and all results collected during the verification effort. This presented a number of limitations, not the least of which was the lack of a central, multi-user repository of verification data.

With the availability of database orientated tools and commercially available VPM solution that utilizes a client-server architecture, many new integration opportunities become evident. This new breed of architecture defines, implicitly, a central server storing MDV data (pass/fail, error/warning signatures, logfile and coverage data locations, among many others). The central server for verification data now aligns this data with other enterprise applications directly related to the verification effort.

Almost all enterprise applications now provide industry standard APIs for access the data they manage. The Representational State Transfer (REST) standard interface definition has emerged as the defacto standard in this space [2], and is broadly implemented, enabling direct access to create, read, update, and delete data via HTTP. REST enables users and other applications direct access to data stored on enterprise application servers. The verification management server, similarly provides a REST API to access MDV data.

REST is a simple way to organize interactions between independent systems. It's been growing in popularity since 2005, and inspires the design of services, such as the Twitter API. This is due to the fact that REST allows you to interact with minimal overhead with clients as diverse as mobile phones and other websites. In theory, REST is not tied to the web, but it's almost always implemented as such, and was inspired by HTTP. As a result, REST can be used wherever HTTP can.

The alternative is building relatively complex conventions on top of HTTP. Often, this takes the shape of entire new XML-based languages. The most illustrious example is SOAP. You have to learn a completely new set of conventions, but you never use HTTP to its fullest power. Because REST has been inspired by HTTP and plays to its strengths, it is optimal for our web and internet orientated business systems.

IV. Defect Tracking Integration

Defect or bug tracking is often one of the highest on users lists of desired enterprise integrations with MDV data. Defect tracking is most commonly implemented as a database application in which users interact with the DB via a standard client such as a web browser. The data is primarily string based, tracked by a simple numbering system, often with the ability to sort by various standard keywords, such as project name, platform type, and so on. While simple in concept, one finds that the application of this simple concept varies wildly across semiconductor companies and even across various design teams within the same company. This gives rise to a key requirement of any integration: it must be highly configurable and flexible. Part of this flexibility must be the ability for data to flow in both directions, that is from the MDV application to the defect tracking application and vice versa.

Beyond these high level requirements, the more nuts and bolts requirements of the integration will vary from team to team, but many user flows will be common. Examples of these flows will include opening a defect report based on a test (simulation, formal or emulation) failure (or group of failures), associating sets of failures with existing defect reports, or indicating to the MDV application which defect reports are designated as closed or resolved. The primary benefit of the integration for these flows emerges as data consistency across the applications (no cut and paste errors) as well as the clear relation of actual test data (pass/fail stored in the VPM application) to defects.

Outside of simply enabling efficiency through data consistency, it’s relative straightforward to see that the ability to sort and filter based on well connected data can enable automation. If a defect is marked as resolved, all of the tests (again simulation, emulation, formal, etc) associated with that defect can be automatically rerun. If a test marked as being associated with a specific defect starts failing (again), a defect state could be updated to reflect this (e.g. the state moved from resolved to open).

V. Enterprise Integration Proof of Concept - Bugzilla

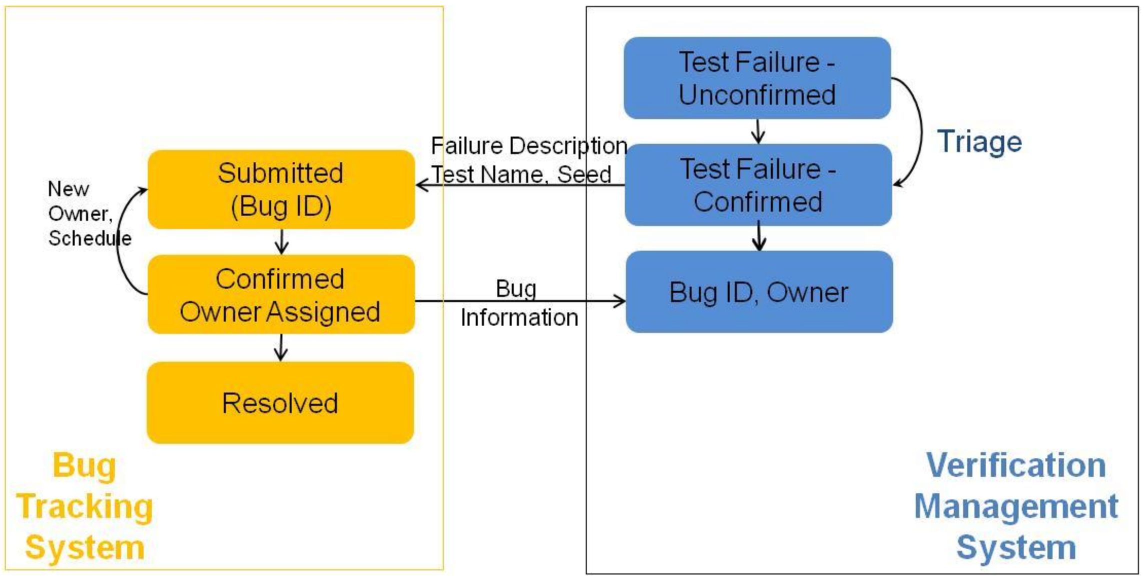
As a starting point for enterprise integration, the Incisive® vManager™ [3] Product Expert Team at Cadence Design Systems, Inc. have developed a proof of concept integration between vManager and the Bugzilla [4] defect tracking system. Bugzilla was chosen because it's freely distributed under the Mozilla Public License, it's easy to install, fairly widely used, and provides a rich set of APIs. It also has excellent support in the open source community, with integration libraries available for almost any programming language.

The proof of concept (POC) initially enabled those first order integration flows mentioned above (open/close a defect, add new tests to an existing defect). The POC also focused on user initiated activity, thus the POC provides efficiency primarily through data consistency and single data entry point (no need to cut and paste test data from vManager to Bugzilla).

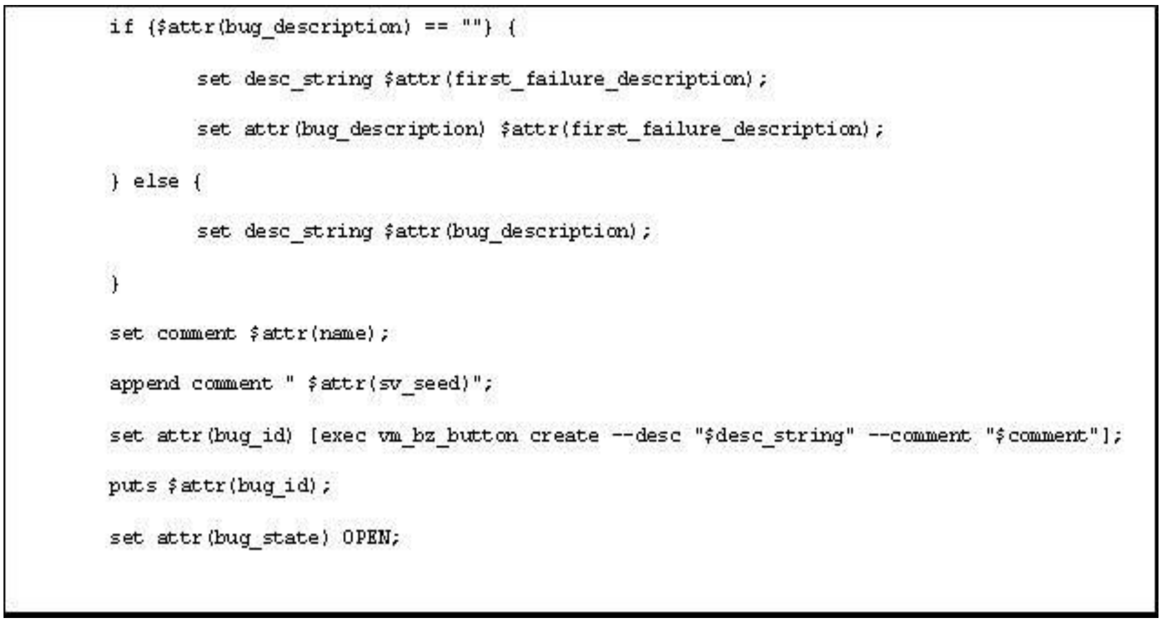
The integration is implemented via the vManager User Defined Action capability, which allows users to write TCL scripts (the scripting interface provided by vManager) which are activated by clicking a user defined button on the vManager GUI toolbar. The user defined button is also related to a data selection in the vManager GUI, such that the test record(s) that are selected by the user are passed to the script. The TCL script associated with various vManager GUI buttons (Create Bug, Append Test to Bug, etc) does a small amount of data parsing, and then calls a Python script to interact with Bugzilla via the Bugzilla REST API. The choice of Python was somewhat arbitrary, as TCL can call any Unix command, and as previously mentioned, many generic REST (and Bugzilla specific) API packages exist in many programming and scripting languages.

The POC enables a number of discrete operations, and describing all of the various combinations, permutations and options in this paper isn't worthwhile. Therefore two common use cases are described.

1) Triage Based Trigger of Bug Tracking Synchronization Flow. To create new defect report, the user selects one or more test (runs) from the Runs Analysis window in vManager. Typically the user would choose a failing run upon which to create a new defect report, but any run record can form the basis of a defect report. After selecting the desired run, the user simply clicks the "Create Bug" button and a new defect report is created in Bugzilla. The defect ID of the newly created report is written to run record in vManager. vManager automatically sets the defect summary (title) to the value of the "First Failure Message" recorded for the run by vManager. vManager also copies the name of the test, and the random seed used in the failing run to the bug report. The integration has the capacity to copy any attribute tracked on a per run basis by vManager into the defect report. Examples of extended data that could be written to the defect report include location of the run logfile or waveform information on disk, the host on which the test ran, and the start and end time of the run. See diagram:

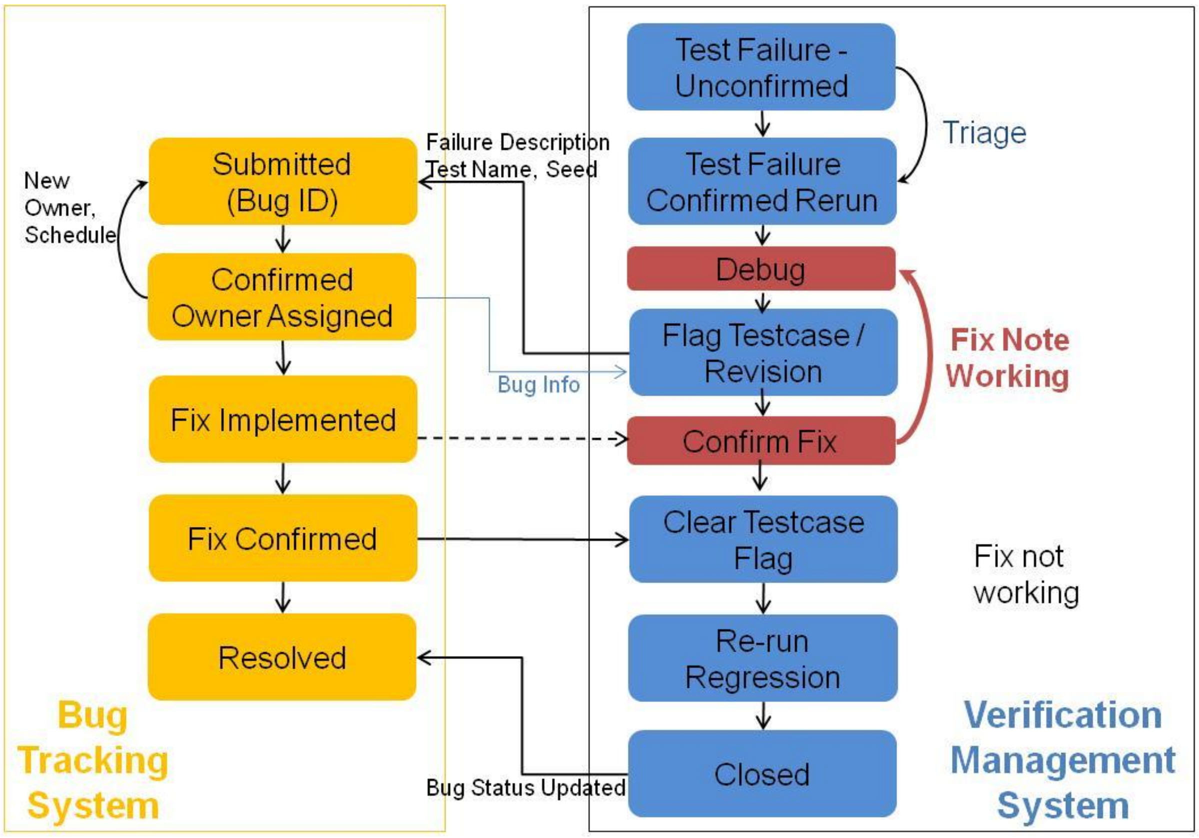


Shown below is the TCL source required to call the Bugzilla API (vm_bz_button) script and pass data to/from vManager. This TCL code can be utilized for either interacting with Bugzilla thru a comma separated value (CSV) interchange, or can be used to interface utilizing the REST API discussed earlier.



```
if {$attr(bug_description) == ""} {

        set desc_string $attr(first_failure_description);

        set attr(bug_description) $attr(first_failure_description);

} else {

        set desc_string $attr(bug_description);

}

set comment $attr(name);

append comment " $attr(sv_seed)";

set attr(bug_id) [exec vm_bz_button create --desc "$desc_string" --comment "$comment"];

puts $attr(bug_id);

set attr(bug_state) OPEN;
```

2) Debug Based Trigger for Bug Tracking Synchronization Flow (Figure 2). For many users, new bugs are not openned until after a more advanced debug process occur. In this case, the user would not open a new bug ID until after the root cause has been determined by additional debug tools. It is commonplace that a specific defect might manifest itself in the failure of a number of runs. In this case, it’s not interesting to open a defect report for each individual failure, but rather to append the properties of the additional failures to the defect report (and report the existing defect report number to the run information in vManager). To perform his task, the user selects a run from the Runs Analysis window in vManager, and clicks the “Append Bug” button. The user is then prompted for the existing defect number. A lookup in the Bugzilla database is peformed, and the ID (if valid), status and summary (title) are populated in the vManager database. Additionally, the name and random seed of the failing run are appended to the defect record. Optionally, users can set a flag in vManager to not execute regressions for a design under test (DUT) which is being fixed, thus optimizing execution resources. Integrations such as this would entail marking that a version or release of a design and testbench combination is in “debug” mode, and additional integrations into Source Control Management (SCM) [5] would be necessary to manage this more detailed flow. Such a marking could indicate to the regression flow to not exercise any regressions on that version. The diagram below represents this more advanced integration of a bug tracking system, a verification planning and management system, and a debug environment – and the necessary flow between them.



3) The next step planned for development is to enable some second and third order integrations with Bugzilla. Some further straightforward scripting should allow the user to extract all of the failures associated with a specific defect ID, and present a report upon resolution of a defect. The verification planning and management tool's integration with job distribution tools such as LSF will then allow the user the ability to automatically re-run all failures, with identical parameters as the failing run, to determine if the defect has been corrected for all failing cases. The POC integration could be further extended to include metrics (coverage) information as well, with the ability to correlate failing tests with a specific section of the verification plan, such that reports on affected features could be generated, or further exploratory tests could be launched.

VI. Extending Integrations Beyond Defect Tracking

Clearly enterprise applications are not limited to simply defect and bug tracking. Connecting to requirements management, revision control, and applications for various flows related verification and the application of MDV will continue to enable greater efficiencies and effectiveness. Beyond these relatively obvious integrations lies the opportunity for analytics that move beyond transactional type operations into more abstract data analysis. One example is the analysis of coverage information collected for a specific set of regression failures associated to a specific defect report. This coverage correlation could then be exploited to seed further regressions that focus on the specific coverage areas to attempt to flush out more defects in a specific solution space.

VII. Scaling Integrations Beyond Point to Point

The REST API mechanism described in this paper is a point to point style information exchange between a consumer and a producer. As the ecosystem of applications that would benefit from connection to VPM platforms like vManager grows, the need may arise to provide connectivity greater than point to point or one to one. Once many enterprise tools are integrated to the design and verification environment, the management and upkeep of the data synchronization becomes a task unto itself. This is not a new problem however, and something that others have already solved using application lifecycle management (ALM) tools, specifically designed to support and maintain enterprise integrations. [6] A number of viable, commercial solutions such as Integration Manager from OpsHub Inc., [7] allows a bus-type architecture and synchronization controls and connectivity between various enterprise tools.

OpsHub is a provider of Application Lifecycle Management (ALM) integration and migration solutions for application development organizations. OpsHub creates a unified ALM ecosystem by seamlessly combining individual ALM systems, enabling agility at scale. A unified global ALM system allows the cross-functional teams using disparate tools in an application development organization to effectively communicate and collaborate with each other, thus increasing overall team agility, productivity and efficiency.

The OpsHub solution enables quick migration and seamless integration between leading ALM systems. The OpsHub solution provides a comprehensive out-of-the-box integration and migration solution within the ALM ecosystem. Its span across ALM functionality, includes requirements management, source control, bug tracking, test management, release management, and customer support.

The integration of VPM tools into such a bus, allows a single adapter to the central bus application to be developed, and enables connectivity to the entire ALM toolset across multiple vendors. This ALM infrastructure is also not limited to Enterprise Tools, but may be a connectivity mechanism between different vendors EDA tools, opening up new possibilities to the EDA world.

VIII. SUMMARY

The design and verification community within EDA are continuously searching for new ways to improve productivity of the process's they must manage. Within semiconductor suppliers, Enterprise systems can no longer exist as separate islands of information, especially when those systems are an integral process to the design and verification (DV) process. This paper sets the stage and defines the strategy for how enterprise systems can seamlessly integrate into the DV process, without requiring users to establish a number of one off and ad-hoc integrations. This paper also provides a proof of concept of such integration utilizing a standardized API framework, such that users do not need to learn custom API's, and can focus their attention on improving their silicon application.

REFERENCES

[1] H. Carter, S. Hemmandy, "Metric Driven Verification" An Engineer's and Executive's Guide to First Pass Success © 2007 Springer Science+Business Media, LLC
[2] http://en.wikipedia.org/wiki/Representational_state_transfer
[3] http://www.cadence.com/cadence/newsroom/features/Pages/vmanager.aspx
[4] http://www.bugzilla.org/
[5] http://en.wikipedia.org/wiki/Source_Control_Management
[6] http://en.wikipedia.org/wiki/Application_lifecycle_management
[7] http://www.opshub.com/main/