A METRICS System for Design Process Optimization

Andrew B. Kahng
UCSD CSE and ECE Depts
La Jolla, CA, USA
abk@ucsd.edu

Stefanus Mantik
UCLA CS Dept.
Los Angeles, CA, USA
stefanus@cs.ucla.edu

Abstract

With shorter design cycles and time-to-market windows, it is essential for designers to have tools and environments that can help them to design more effectively and efficiently. While some design time may be wasted due to incorrect use of resources, other time may be wasted due to inefficient *design process*. Our work addresses the fundamental issues of understanding, diagnosing, optimizing, and predicting the system design process. Work in [1] described a prototype system called METRICS that allows automatic recording of design process information (see Figure 1 for METRICS architecture). METRICS comprises a web-based architecture based on industry-standard components (HTTP, RDBMS, XML, Java servlet, etc.) and standardized tool metrics (naming and semantics); its purpose is to enable automatic and fair comparisons of both design instances and design optimization results. METRICS (i) unobtrusively gathers characteristics of design artifacts, design process, and communications during the system development effort, and (ii) analyzes and compares that data to analogous data from prior efforts. Specifically, the infrastructure consists of (i) a standard metrics schema, along with metrics transmittal capabilities embedded directly into EDA tools or into wrappers around tools; (ii) a metrics data warehouse and metrics reports; and (iii) data mining and visualization capabilities for project prediction, tracking, and diagnosis.

In [2], we extended the METRICS system to allow optimization of design processes at the flow level, rather than only at the individual tool level. The updated infrastructure includes collection of design flow information for use in flow optimization, as well as integration with datamining tools to allow automatic generation of design and flow quality-of-result (QOR) predictors.

This presentation describes the implementation and deployment of a METRICS system at UCLA

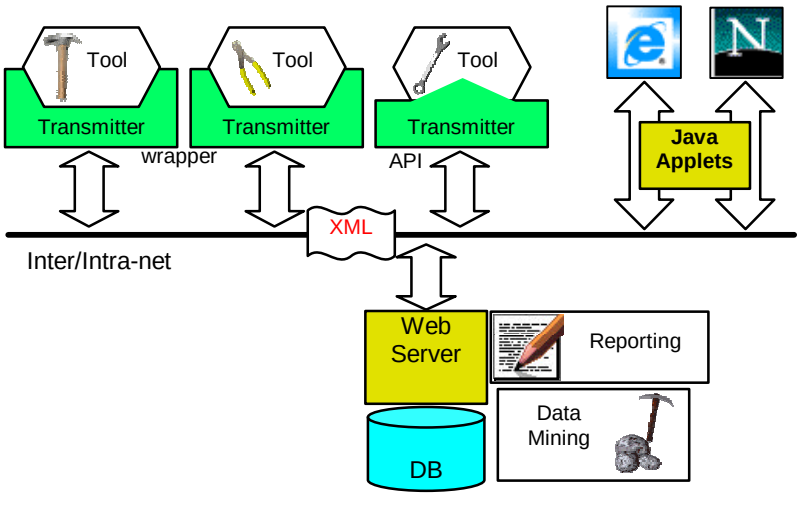

Figure 1. METRICS Architecture.

and in several industries networks, which we have used to collect tools information (e.g., from Silicon Ensemble regression tests). With the *Cubist* datamining tool, we have constructed predictors of various aspects of solution quality for the QPlace, CTGen and Warp Route tools. In other efforts, METRICS has been extended to capture more tools and design flows including SPW, SP&R, and Verification. The current METRICS system uses thin Java wrappers around tools for capturing design informations. The METRICS server is implemented using a three-layer architecture that consists of (1) a web server and Java servlets as the top layer for accepting report requests and design information, (2) Java beans as the middle layer that is used for processing both the reporting and the transmitting of design data, and (3) an Oracle database as the bottom layer for storing all collected design metrics. Experiences obtained from this deployment of the METRICS system include such issues as the following.

Tool communications are always a big issue that must be handled when integrating several tools together. When writing tool wrappers, we will always face problems because each tool is developed differently than the other tools: there is no common way for either passing controls to the tools or retrieving information from the tools. For example, some tools use configuration files to set the run options while others require use of a scripting language for the same purpose. Similarly, design information can be collected from a single log file for some tools while for others, the analogous information is spread around several different log files and report files.

Security of design data may be another issue. For example, if the METRICS system is deployed inside an Application Service Provider (ASP) site, on which several different customers are using the ASP's services for designing their products, security is paramount for any metrics collected inside the ASP. When transferring data from the tools to the database, the data must have appropriate levels of encryption and other safeguards.

Effects of wrappers on wrapped tools is another important issue. Overhead of wrappers should be as small as possible. In addition, the robustness of the tools should not depend on the robustness of the wrapper, i.e., if a wrapper fails, the tool that it wraps should run normally. Furthermore, in the event of a network problem that prevents the transmission of data to the METRICS server, the wrapper must be able to store metrics data locally without halting the execution of the tool.

Integration of the METRICS system with license managers is important not only because this is an easy path to useful data (e.g., by tracking license activity we can obtain some indication of when we need to add licenses to the CAD resource), but also because METRICS needs to be integrated with other process and resource management tools such as Platform Computing's Load Sharing Facility (LSF).

Finally, the METRICS system must also be able to **report bugs or other problems/exceptions that occur when running the tools**. Tools designers can benefit from METRICS reports that identify runtime configurations that produce errors. Semi-automated diagnosis of tool "sweet spots" (field-of-use, and proper configuration) is also possible, e.g., recent experiments with running variant PKS flows allowed us to assess the utility of wire load models (WLMs) for timing optimizations.

Reference:

[1] S. Fenstermaker, D. George, A. B. Kahng, S. Mantik and B. Thielges, "METRICS: A System Architecture for Design Process Optimization", *Proc. ACM/IEEE Design Automation Conf.*, June 2000, pp. 705-710.

[2] A. B. Kahng and S. Mantik, "A System for Automatic Recording and Prediction of Design Quality Metrics'", *Proc. Intl. Symposium on Quality in Electronic Design*, March 2001.