

Cavity Data Management System Proposal (draft 3)

1/10/2007

Jerzy Nogiec

Marc Paterno

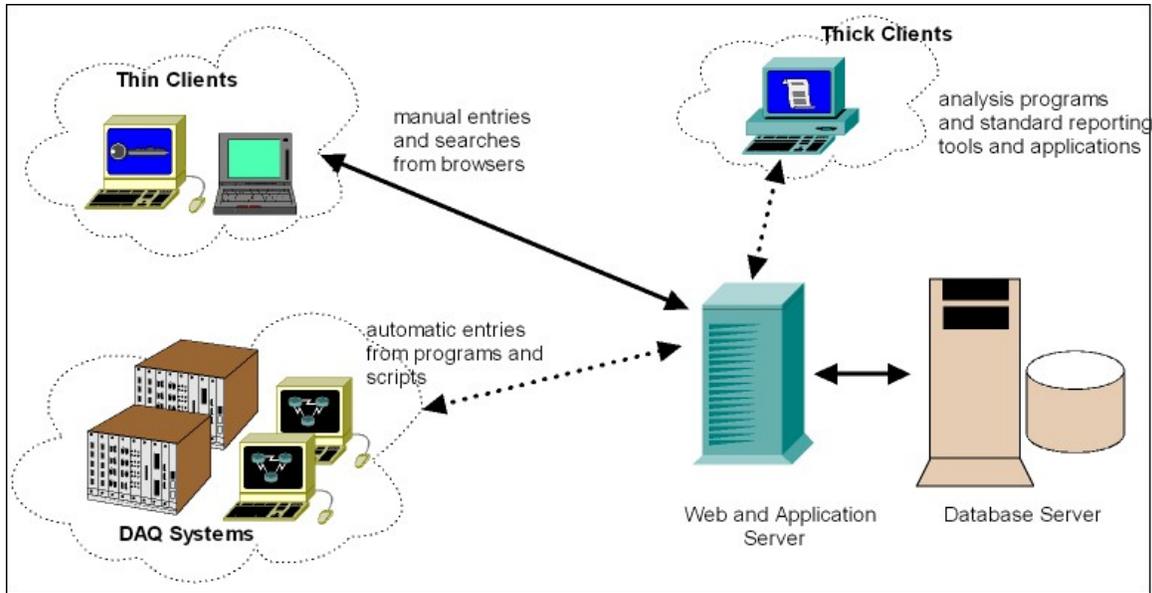
Claude Saunders

The following document is a proposal for developing an in-house web and database-based system for managing data produced during cavity production and testing. The focus is on accommodating rapid changes in cavity production and testing processes. The key concern is balancing the needs of flexible data entry with the stability of standardized reports.

1. Functionality

- a. See Master requirements "ILC Data Management System Core Functionality Rev 2.0".
- b. Roles
 - i. Administrator
 - ii. Process Engineer (designs traveler types)
 - iii. Process Manager (approves traveler types)
 - iv. Test Manager (assigns travelers)
 - v. Tester (completes assigned travelers)
 - vi. Analyst-Contributor (uploads comments, analysis results)
 - vii. Viewer Non-contributor (browses through data, reads reports)
 - viii. Software and DB maintainer
 - ix. Integrator (develops applications integrated via web services)
- c. The development effort is intended to be agile and incremental in nature. We therefore propose to implement only a subset of the total requirements above initially. With the first implementation, the user (in some cases, a restricted set of users) will be able to perform the following tasks:
 - i. Create a new traveler type via web interface.
 - ii. Preview, approve, and deploy new traveler types, using a workflow that implements whatever procedure is required for the approval process.
 - iii. Create a new traveler instance containing steps, instructions, and data fields.
 - iv. Populate data fields during cavity production and testing via web interface only. Automatic uploading of data shall not be possible with the initial version.
 - v. Create login ids and roles, and associate roles with login id (this function is restricted to administrators).
 - vi. Login to the system using some authentication scheme, obtaining the authorized roles assigned to the login id.
 - vii. Modify a traveler type (as opposed to creating a new type), retaining versioning information that associates the new version with the older version(s) of that traveler.

- viii. Produce one of a small number of reports on traveler instances. In the initial version, these reports will be available in web page format only.
 - 1. Show all traveler instances for a given cavity id (or more generally, entity id (*i.e.* half-cell, clean-room, *etc.*)).
 - 2. Show all traveler instances of a given traveler type.
 - 3. Show all attribute names and values for a given traveler instance (data view of traveler instance).
 - 4. Show traveler instance with steps and instructions (text view of traveler instance, also suitable for printing).
 - 5. Show in-progress/completed traveler instances.
 - ix. Export the data from the reports above in CSV format.
 - d. In addition to the human-centered functionality above, the initial implementation will contain a prototype web service which provides an interface to obtain traveler data. This will be used to verified compatibility with a commercial reporting tool (*e.g.* Crystal Reports).
2. Architecture
- a. Multi-tier architecture
 - i. client tier,
 - ii. web application and service tier
 - iii. back-end systems including databases and other data sources
 - b. Connectivity
 - i. Other systems and applications that need to be integrated (XML-based web services).
 - ii. Thick clients. These may include specialized analysis programs and standard reporting tools/applications.
 - iii. Thin clients such as web browsers or PDAs to create and complete travelers (HTTP).
 - iv. On-line data acquisition and measurement systems (XML-based web service vs. direct database access vs. file upload and processing in the application server) (?)



3. Technology

a. Conventional web site vertical:

- i. Web browser (for client)
- ii. Apache web server for external authentication mechanism. We are considering using *Central Authentication Service* (CAS, see <http://www.ja-sig.org/products/cas>).
- iii. Tomcat Servlet Engine, including use of
 1. Java Server Pages (JSP: <http://java.sun.com/products/jsp>)
 2. Java Server Faces (JSF: <http://java.sun.com/javaee/javaserverfaces>) or Struts (<http://struts.apache.org>) or both.
 3. Java Database Connectivity (JDBC: <http://java.sun.com/javase/technologies/database/index.jsp>)
 4. Java Data Objects (JDO: <http://java.sun.com/products/jdo/>)
- iv. Oracle Database
 1. Conventional table design with dynamic addition of columns to support evolving traveler definitions.
 2. Metadata tables to drive web-based interfaces.

4. System Interfaces

a. Main functional web interface categories (human).

- i. Administrative
 1. define system users
 2. define system roles and assign to users
- ii. Entity Management
 1. Create new entity (cavity, clean-room, coupler, half-cell)
 - a. *ie.* some physical thing to which a traveler type would be primarily associated.
 2. Manage entity namespace (*e.g.* Allowed cavity id names)
- iii. Traveler Type Management

1. Create Traveler Type
2. Modify Traveler Type
3. Preview
4. Approval
5. Deployment
6. Browse existing types
- iv. Traveler Instance Management
 1. Create new instance
 2. Executing steps of instance (acknowledging steps and/or entering required data)
 3. Print traveler for manual use (and subsequent data entry)
- v. Traveler Instance View
 1. Browse instances (clicks)
 2. Query instances (*e.g.* enter cavity id or traveler type to find all)
 3. Export to the CSV format
- b. Web Services
 - i. Intended for all eventual reporting access and application integration.
 - ii. A stable interface to a potentially changing underlying representation (*e.g.* Evolution of traveler type).
 - iii. May support specific queries.
 - iv. May support general data discovery and access for general purpose reporting and analysis tools.
5. Effort Breakdown
 - a. Technology proof-of-concept
 - i. Development framework
 1. set up subversion repository, provide Kerberos access to developers
 2. provide central Postgres instance for developers, with individual accounts and individual working space
 3. each developer must have access to the following tools:
 - a. Java development kit (JDK)
 - b. Tomcat
 - c. subversion (svn) client
 - d. JDBC driver
 - e. JSF implementation
 - f. Struts implementation
 - g. Ant implementation
 4. optionally, each developer may want a local Postgres installation; developers may use instead the central development installation listed above
 - ii. Coding patterns
 1. establish development process conventions which allow for use of IDE's, but do not require them

2. establish “hello world” example to distribute as starting point to all developers
 3. establish “nightly build” or “continuous build”, to build and test the code at the head of the repository
 - b. Database Design
 - i. Meta data tables
 - ii. Traveler tables – dynamic, so specify behavior
 - c. Web Interface Buildout
 - i. Administrative
 1. 6 pages
 - ii. Entity Management
 1. 4 pages
 - iii. Traveler Type Management
 1. 17 pages
 - iv. Traveler Instance Management
 1. 5 pages
 - v. Traveler Instance View
 1. 8 pages
 - d. Prototype web service
 - i. Data access
 - ii. Verify commercial reporting tool can work with it.
 - e. Deployment
 - i. Staging/integration framework
 1. set up central staging oracle instance
 2. set up internal-only staging instance of production framework
 - ii. Production framework
 1. set up apache
 2. set up tomcat
 3. set up production oracle instance
 4. set up file repository
 5. set up Central Authentication Server (CAS)
 6. establish backup/restore policy
6. Deployment Considerations
 - a. Central Oracle
 - b. File System for uploaded content (PDF, JPEG, *etc...*)
 - c. Hosts for Apache and Tomcat (externally accessible)
 - d. Authentication source