TIssue v2 Roadmap

Lauri Carpenter

Randy Reitz

Overview

- High-level Goals
- More Detailed Map
- Timelines

High-Level Goals

1. Layered, modular architecture

   This will de-mystify detector writing!
2. Exemptions
3. Code maintainability/deployability
4. Unit tests
5. Eliminate Zope in favor of less heavyweight web tools
6. Documentation

Goal 1: Layered, modular architecture

- Published list of web services provided by TIssue ("we", SOAP service)
  - Event_Factory methods
  - Exemption methods
  - BlockEngine methods
  - … in fact, ALL access to DB through this SOAP service
  - Clients should not ever do direct SQL ("select…", "insert…", etc.)
- Many customers ("they")
  - Detectors
  - Scanners, inventory refreshers, report writers
  - CLI access (sooner)
  - GUI access (later)
- All detectors use the same generic methods
  - No more need for a new server for each detector!
- Detector-specific information handled via configuration
  - Keyword/value configuration parameters

Goal 1 (cont'd)

- De-Mystify Detector Writing
  - Detector (customer):
    - determines when an "event" has occurred
    - knows the specific event "details"
    - sends the event information via one web service call
  - Server:
    - Handles "event" based on:
      - Detector configuration
      - Event details
      - Exemption information

Goal 2: Exemptions

- Exemption type determines who is interested
  - Etype SCAN: scanners
    - Replaces hard-coded lists inside of multiple scripts
  - Etype ISSUE: event creation
    - Replaces hard-coded lists inside of multiple scripts
  - Etype WHITELIST: blocking engine
    - Becomes data-source for blocking engine whitelist file (maintained separately as a fail-safe)
- Common data source for all exemptions – now we can generate reports!

Goal 2 (cont'd)

- Common exemption request interface
- Add "work-flow" for e-signature approval
  - Re-use existing plone mechanism unless another work-flow engine becomes available

Goal 3: Code maintainability/deployability

- Separately deployable component packages
  - Not one great big steaming pile!
    - Clients will not need Zope, postgres_client, oracle_client, mod_python, etc. in order to write a detector!
    - Developers will not need to release new scanners when DTML files need updating
  - Use ids_shutil release scripts (see documentation)

Goal 4: Unit Tests

- Include unit-tests in subcomponents
- “make test” becomes part of standard release scripts

Goal 5: Eliminate Zope

- Zope is complicated and heavy
- It would be nice to migrate existing Zope-based pages into django pages
- Not a high priority

Goal 6: Documentation

- Use py_cmd_util for consistent self-documenting commandline/python interface
- Code the CLI, test the CLI; later, call the implementation from the GUI
- Use pydoc or other documentation-generators for other documentation
- Use “good programming practices” to document the code!

More Detailed Map

1. Package re-organization
2. TIssue v2 DB design (including exemptions)
3. Object-oriented breakdown of components
4. Tissue DB server (web service access to DB)
5. Proof-of-concept Detectors (customer)
6. Blocking Engine
7. Event sweeper (email reminders, auto-close, etc.)
8. Detector configuration interface (GUI)
9. Issue configuration interface (GUI)
10. Exemption system (request GUI, approval GUI, creation GUI)

1) Package Re-organization

- Lots of small self-contained packages that can be separately maintained/deployed
  - Do ONE thing and DO IT WELL!
- Ups table files denote dependencies between products
  - TIssue_config
  - TIssue_common
  - TIssue_event_factory
  - TIssue_ddl
  - TIssue_monitoring_tools
  - TIssue_????

2) TIssue v2 DB Design:

- Use modeling tools (DeZign)
- Changes for detector configuration (to eliminate detector-specific “out-of-band” communication)
- Changes for exemption persistent store
- Changes to allow for mac-based and system-based event creation (currently only ip-based events)
- Optimizations, reduction of miscomp-duplication
  - Question: does Tissue v2 require a separate cache of miscomp-duplicated information, or can it STOP when miscomp is unavailable? Or something in between (queue event until miscomp becomes available?)

3) Object-oriented breakdown of components

- Think about the interactions
- Use a design tool
- Recognize and use design patterns
- Get some expert help!

4) TISsue DB server (web service)

- Single point of communication for customers
- Initial release must include:
  - Event_Factory methods
  - “read exemption” methods for scanners, blocking engine
- Need to design authentication/authorization mechanism
  - Would like some help setting this up!
- Requires TIssue v2 DB design
- Requires new remediation page GUI
  - Can be “cloned” from existing page

5) Proof-of-concept Detectors

- Recode 2 or more existing detectors as proof-of-concept while developing server
  - Linux Baseline
  - Sms_aged_pwd_detector
  - Vscan
  - netVer
  - FCIRT event injector

6) Blocking Engine

- Recode existing blocking engine using new DB interface
- No direct access to DB; instead, call methods from the Tissue dbserver for "get_machines_to_[un]_block" and "update_machine_state", etc.
- This will decouple the blocking engine from the Tissue implementation and db schema

7) Event Sweeper

- Event sweeper handles the scheduling of:
  - Email reminders
  - Auto-close of events
  - Blocking engine requests
- Recode existing interface to use new DB interface
- This will decouple the event sweeper from the TIssue implementation and db schema

8) Detector configuration interface

- Email_template
- From_address, CC_address, ReplyTo_address
- Authorized detector admins (persons allowed to modify detector configuration)
- Authorized detector machines (machines allowed to communicate with event_factory for this detector type)
- Detector-specific keywords, default values
- ??? (to be determined)

9) Issue configuration interface

- Associated detector
- Severity
- Block? (block_delay)
- Send email? (reminder frequency)
- Associated with which (if any) net services?
- Issue-specific keywords
- ??? (to be determined)

10) Exemption System

- Exemption Request interface
  - Requested exemption type?
  - Requested by whom?
  - Requested for which system/cluster?
  - Reason?
  - Duration?

7) Exemption System (cont'd)

- Exemption Request Approval
  - Needs "political" decision of who needs to approve (possibly depending on exemption type)
  - Re-use alphaflow infrastructure unless other generic workflow engine becomes available
  - Role-based, not person-based!

7) Exemption System (cont'd)

- Exemption Creation
  - Interface to create exemptions, possibly pre-populated with data from an exemption request
- Exemption Renewal Process
  - Email reminders/notification
  - Re-approval?
- Reporting Tools

Timeline

- March 2010:
  - DB, OO design review, assistance (hopefully)
- Early April 2010:
  - TIssue v2 db design (DDL)
- April – July 2010:
  - TIssue DB web service
    - Assistance needed for authentication issues
    - Include “read exemptions” interface for scanners, whitelist
  - 2+ detectors using TIssue DB web service
  - Blocking Engine using TIssue DB web service
  - Event sweeper using Tissue DB web service
  - (expert cli interface for detector config, issue config, exemption config; i.e., NO GUI!)
- End of July 2010: Production release Tissue v2 (no GUI)
  - Selected (probably 2) detectors

Timeline (cont'd)

- Later:
  - Migrate more detectors
    - Recode existing inventory refreshers to use new tools
  - Add GUI interfaces:
    - Detector configuration interface
    - Issue configuration interface
    - Exemption request/approval/creation interface
- And monitoring tools…NOT zope-based
- And reporting tools and sparklies… NOT zope-based
- And eliminate Zope…