ABRT Project Document

Karel Klíč

May 28, 2012

ABRT improves quality of operating systems by automatic detection of software failures, their analysis on a centralised server, and reporing failures to the right group of developers. It is designed to help users to report problems, help developers to fix the bugs found in the reports, help security engineers to determine the security impact of a failure, and help quality assurance and release engineering people to assess the overall quality of an operating system release.

This is a top-level project document for ABRT. It describes the top-level design, communication protocols and interfaces. It also describes the current status of the features, and project roadmap.

Contributors. Miroslav Lichvár contributed figures 1 and 2. Miroslav Lichvár and Michal Toman co-designed the database schema for the Problem Storage. Jiří Moskovčák co-designed the use cases. Jon McCann designed the graphical user interface of the client.

Contents

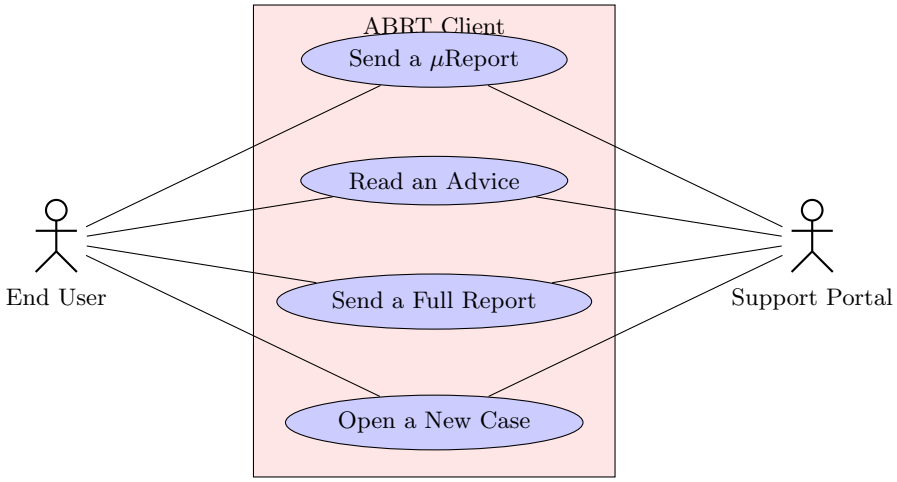
1 Project Charter

1.1 Statement of Work

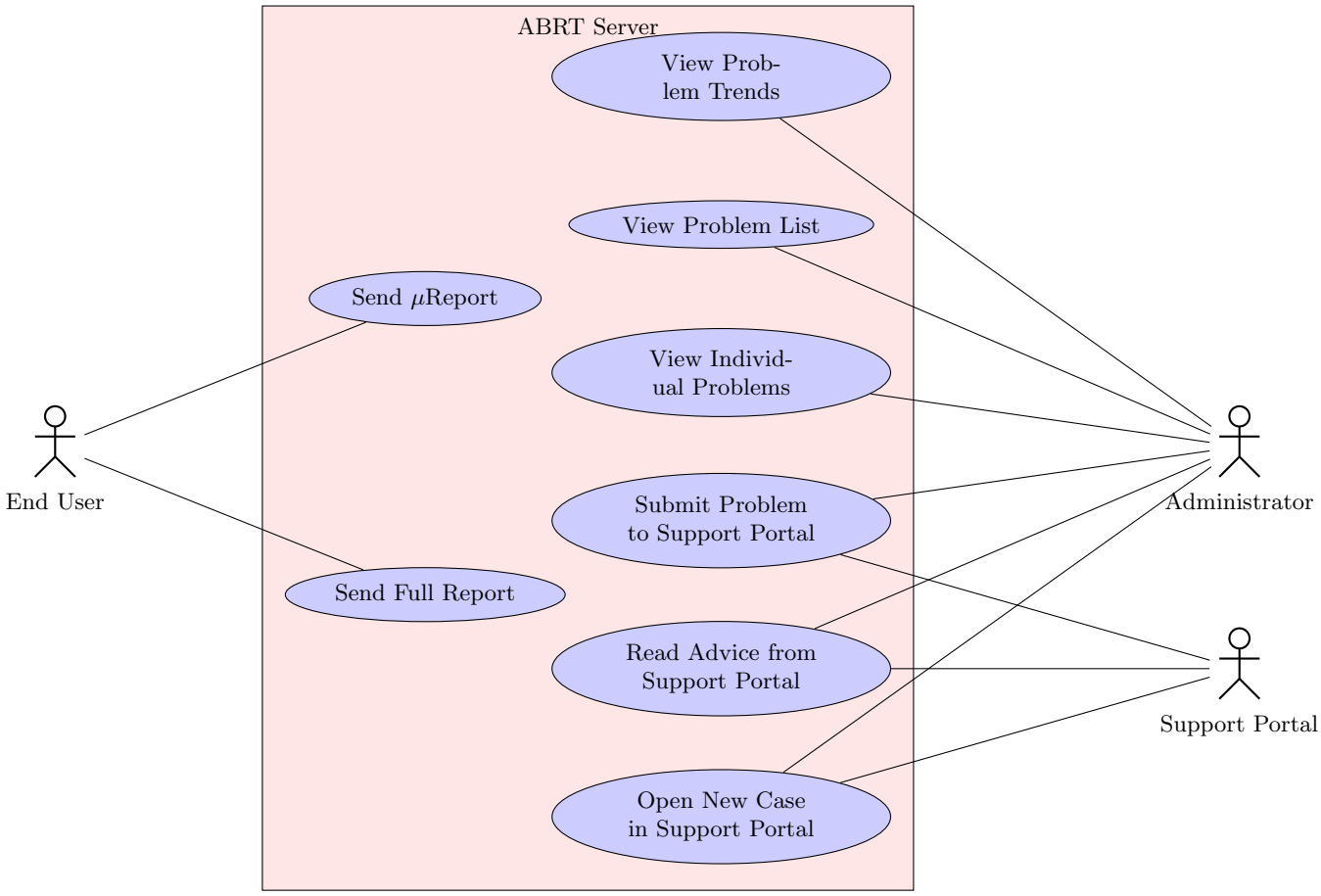
business need, product scope, strategic plan

1.2 Business case

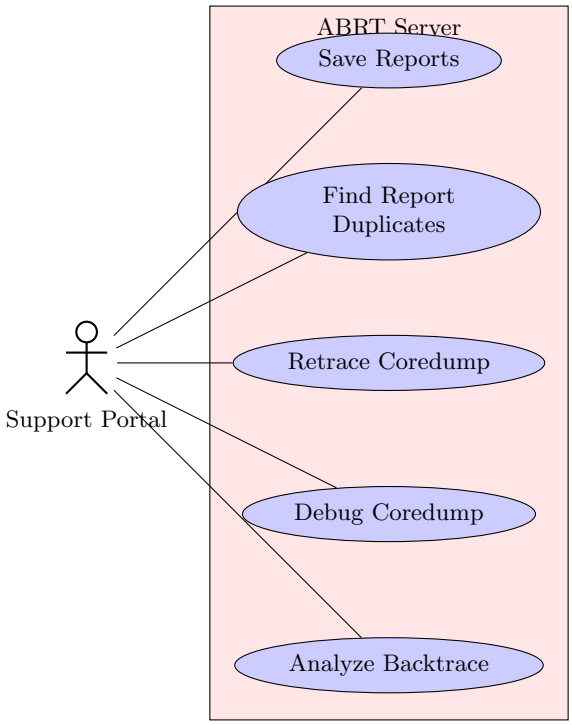
1.2.1 Enterprise Client-Side Use Case 1



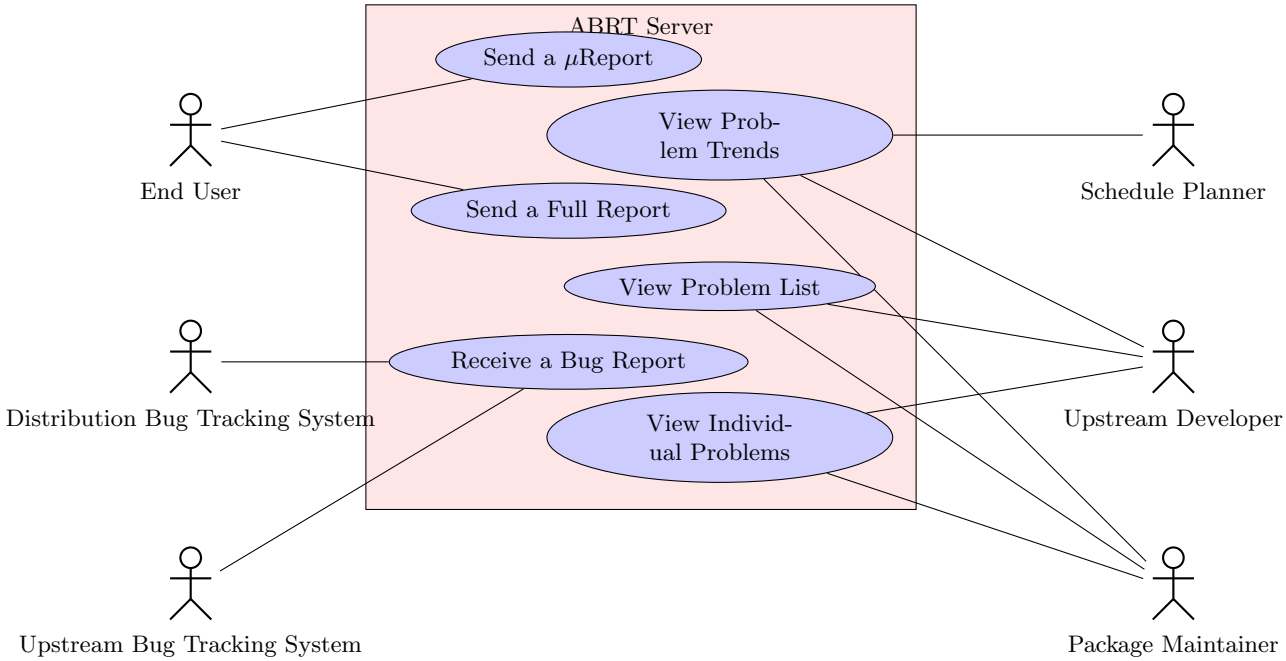
1.2.2 Enterprise Client-Side Use Case 2



1.2.3 Enterprise Support Use Case



1.2.4 Fedora



2 Stakeholders

Here is a list of people impacted by the project, and relevant information regarding their interests, involvement.

Users[3]
- Inform user about the problem and apologize
- Help with recovery of previous state: (automatically recover, explain what to do, offer logout/login or restart)
- Allow user to participate in the improvement of the system

Upstream Developers
- Investigating security-sensitive bugs

Package Maintainers
- Fixing most frequently occured bugs
- Fixing bugs where the bug is well described

Tools Team
- Providing knowledge about ELF, DWARF, GCC, GDB, elfutils
- Implementing extensions to the tools to allow new ABRT features

Support Team
- Integrating parts of ABRT Server into their support workflow
- Server interface for customers

Security Team
- Notified about security-sensitive bugs
- Investigating security-sensitive bugs

Desktop Team
- ...

QA Team
- Watch component problem statistics. Discover regressions and allocate QA attention and effort accordingly.

Stakeholder register
Stakeholder management strategy

3 Requirements

Large-scale data collection ————————-
Collect anonymous small reports
Show quality of operating system - overall quality of a release over time - quality of a single package
Highlight important reports - some reports require fixing soon
In-depth bug fixing ————————-
Collect core-dump for frequently occured reports
Retrace coredumps on demand
Allow interactive support for debugging of coredumps

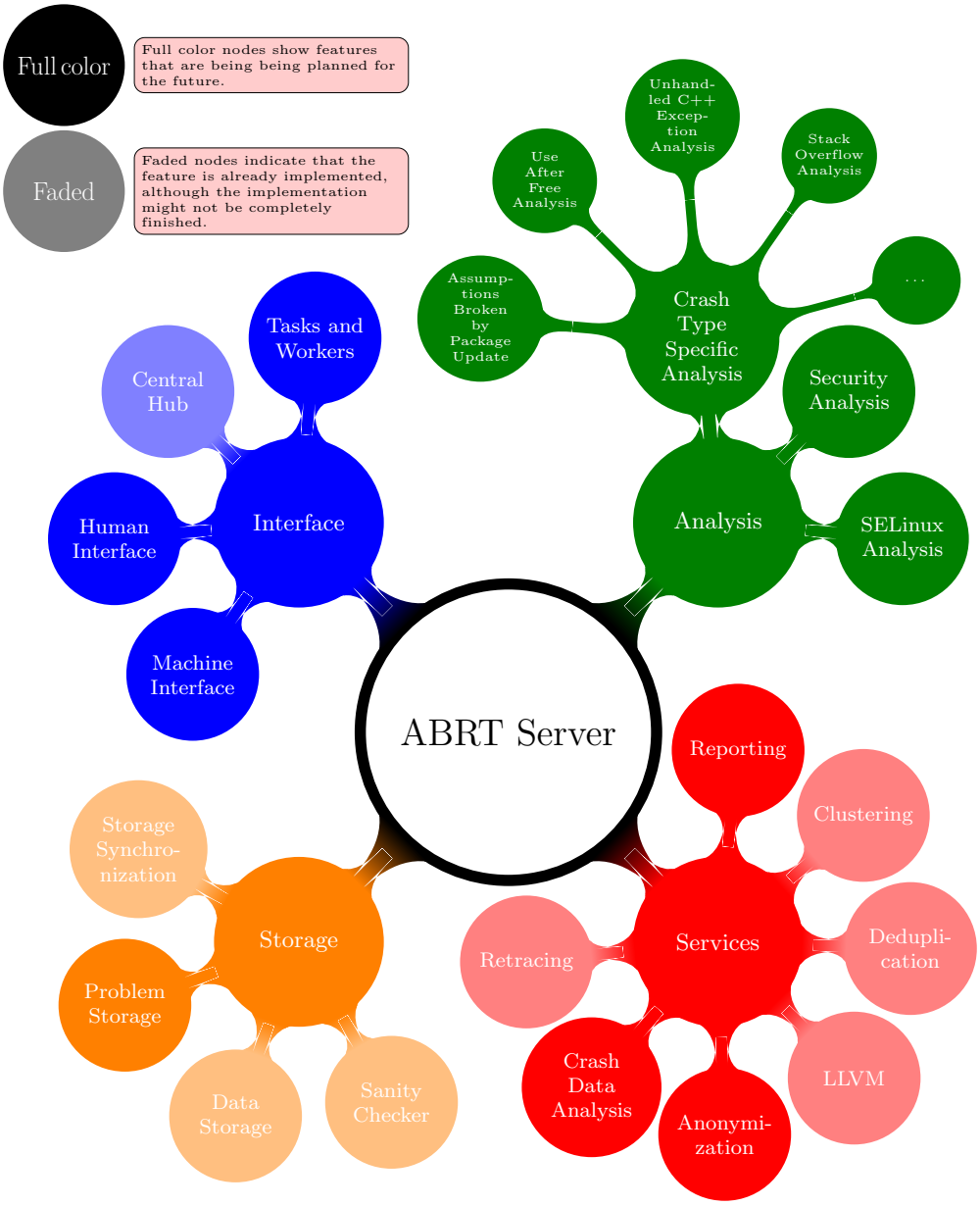
Developer support ——————
Analyze source code around a crash and find the problem
Provide source code browser
Communication ————-
Report only problems with enough information
Report to operating system bug tracker
Report to upstream bug tracker
Watch bug resolution in remote tracker

4 Scope

5 Work breakdown structure

Deliverable oriented decomposition of server project into smaller components.

5.1 ABRT Server Overview



Storage Server's storage is a combination of a database and a file server.

Storage Synchronization Fetches data from external systems, such as RPMs and builds from Koji, bugs, comments, attachments from Red Hat Bugzilla, components, maintainers, releases from Fedora Package Database.

Problem Storage We store all issues reported by users here. Problems can be program crashes, uncaught Python exceptions, Kernel oopses, VM cores, and SELinux denials.

Data Storage We download and store RPMs of all versions of all packages in operating system. This is necessary for correct retracing of both coredumps and minidumps. We require both binaries and debugging information. Static analysis requires data files from RPMs for greater accuracy.

Sanity Checker We measure quality of data (builds, RPMs, bugs) downloaded from Fedora Project. We found quality measurement and defect detection to be necessary to keep any service operational. For example, many packages in Fedora and RHEL lacked proper debugging information, and this issue blocked retracing of many coredumps. We started to track the quality of debugging information and watch for regressions.

Services Separate services working on the top of Problem Storage and Data Storage. They are triggered by creating a new issue, changing the state of the issue, or at a certain time interval.

Retracing Generates full backtrace from a coredump stored in Problem Storage. Generates function names from a minidump (coredump-level backtrace) stored in Problem Storage.

Crash Data Analysis Depends on Retracing.

Anonymization Depends on Crash Data Analysis.

LLVM Depends on Anonymization.

Deduplication Deduplication happens on several levels: minidumps (coredump-level backtraces) are compared when receiving a new issue. Backtraces and/or function names (we call the list of function names from the crash thread *optimized backtrace* in Faf) are compared when Retracing step is done. Analyzed issues are compared when the Crash Type Specific Analysis is done.

Clustering Clustering finds clusters of issues that are close (similar) to each other. Clusters are created fro multiple distances. They are used to determine possible components and even program functions that are the root cause of the bug.

Reporting

Analysis Based on static analysis techniques applied to LLVM bitcode, the Analysis framework investigates reported issues and provide insight at the source code level.

Crash Type Specific Analysis

Security Analysis

SELinux Analysis

Interface World-facing communication is implemented in the web-based Human Interface, and JSON-based Machine Interface. Internal interactions between server parts are organized as tasks performed by workers. Task queue is managed by central hub, which also provides the public communication interfaces.

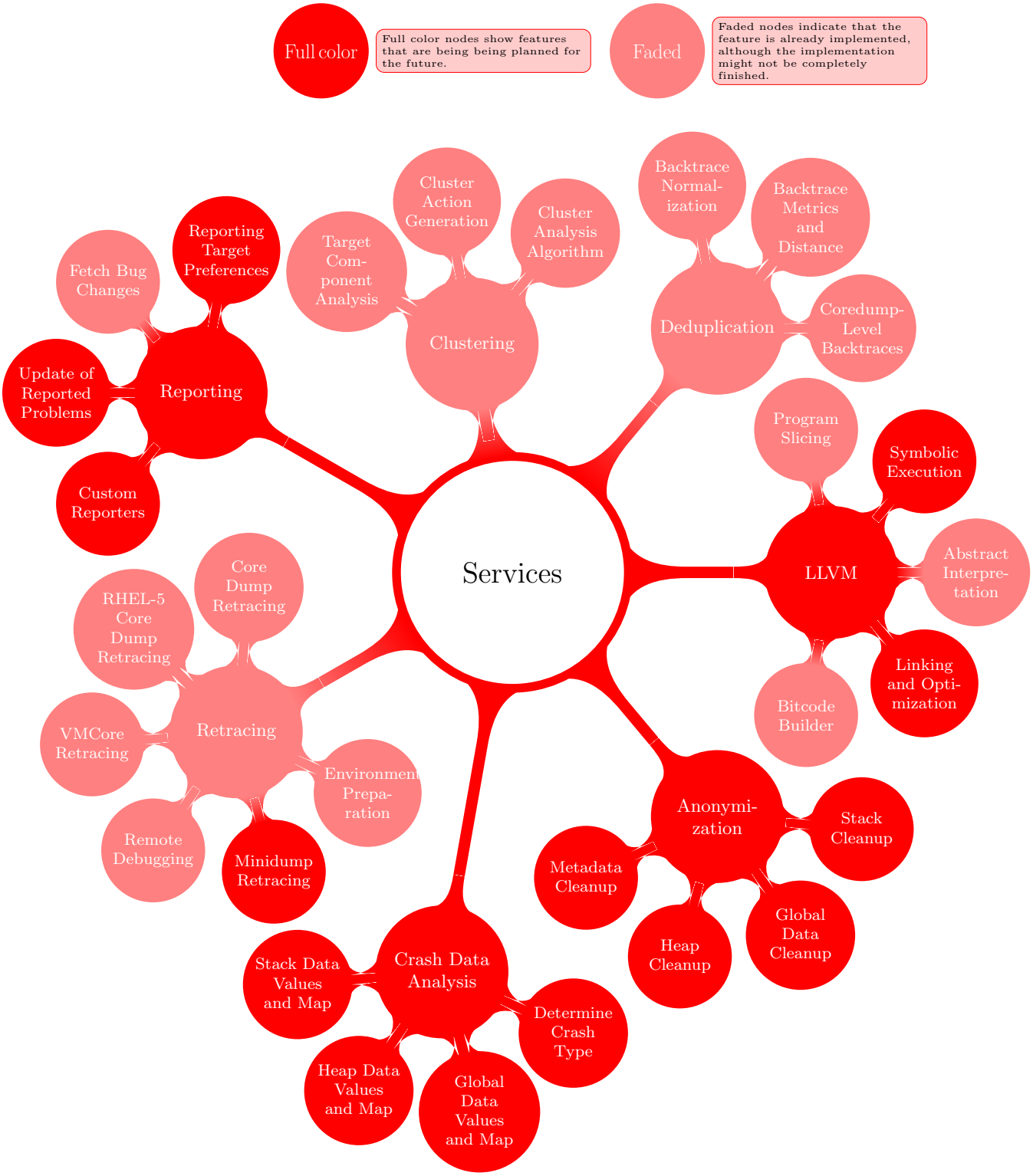
Human Interface

Machine Interface

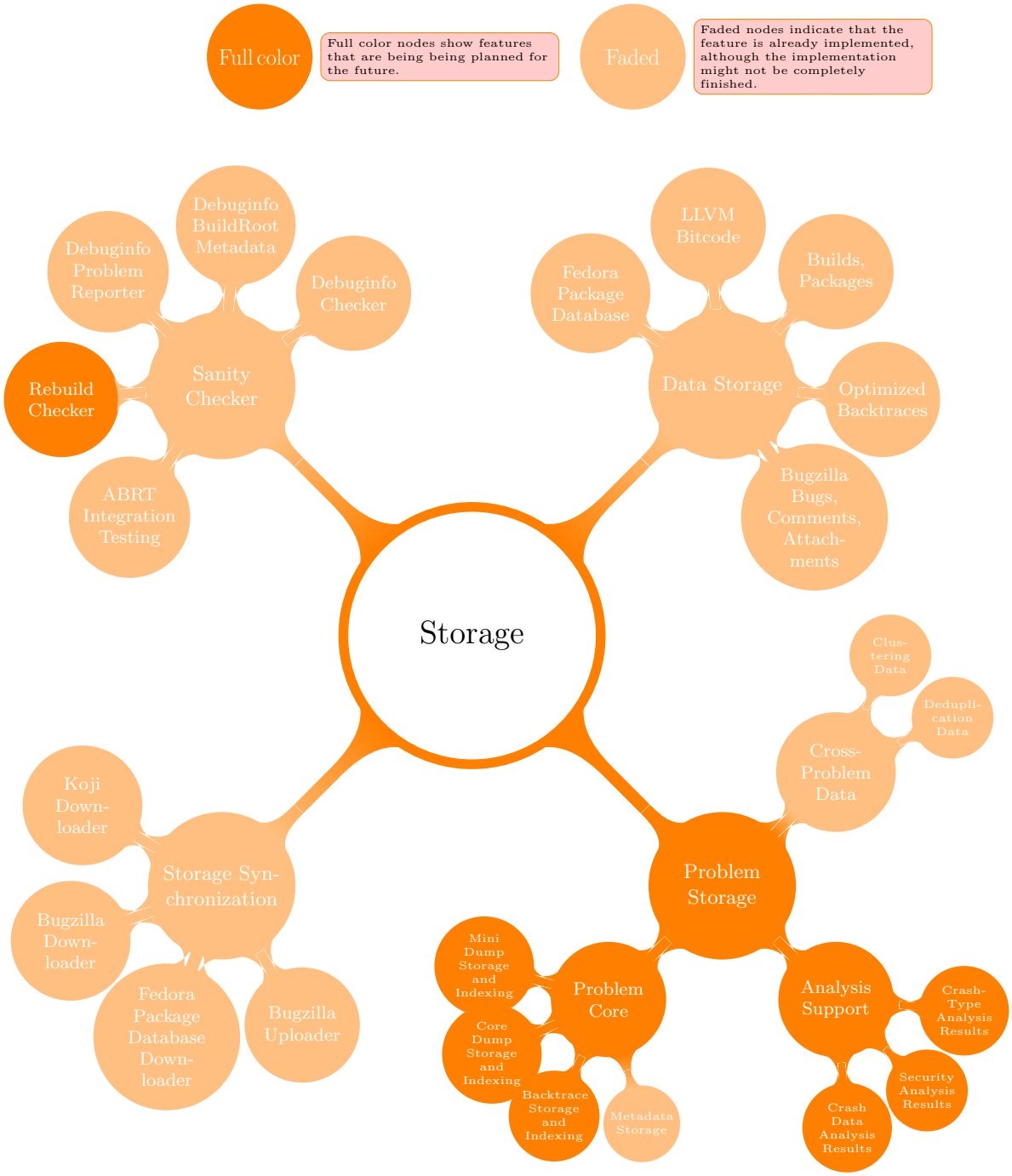
Central Hub

Tasks and Workers

5.2 ABRT Server Services Overview



5.3 ABRT Server Storage Overview



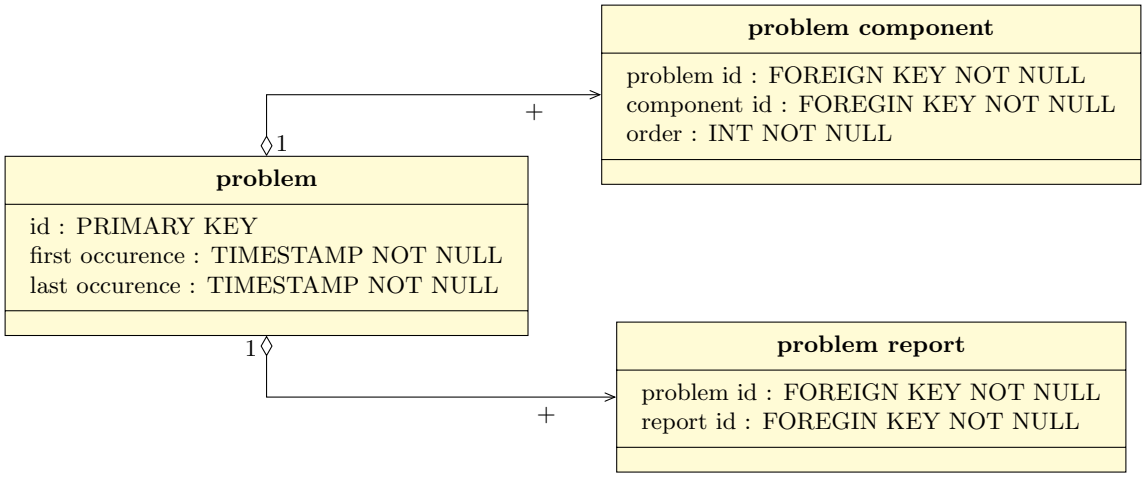
Storage Synchronization Bugzilla Downloader Downloads server-related bug reports from Red Hat Bugzilla.

Data Storage Database and file storage for data required for the analysis, evaluation, adn processing of reports and problems.

LLVM bitcode We store LLVM bitcode of every binary and dynamic library compiled from C/C++

source code.

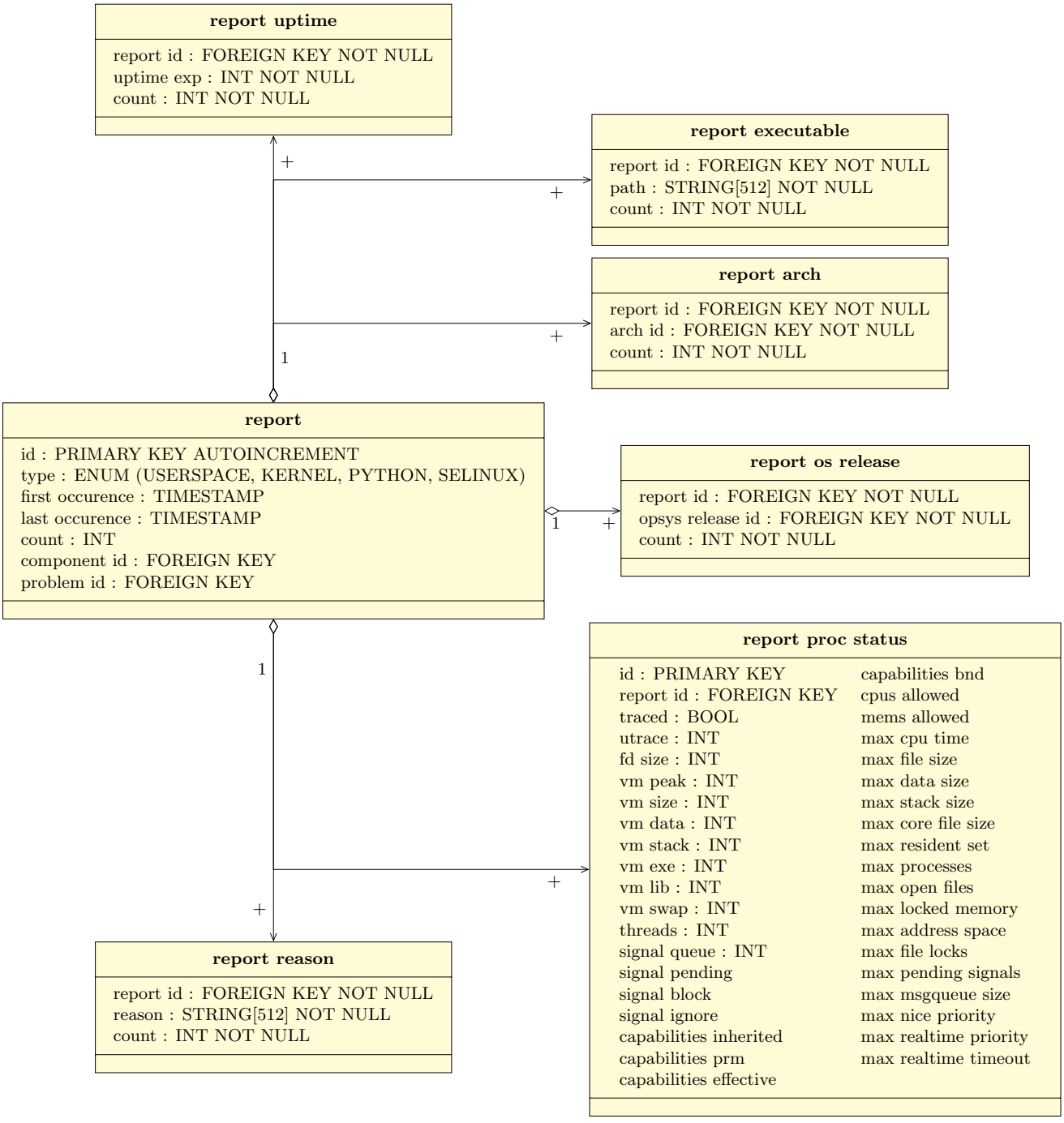
5.3.1 Problem Storage



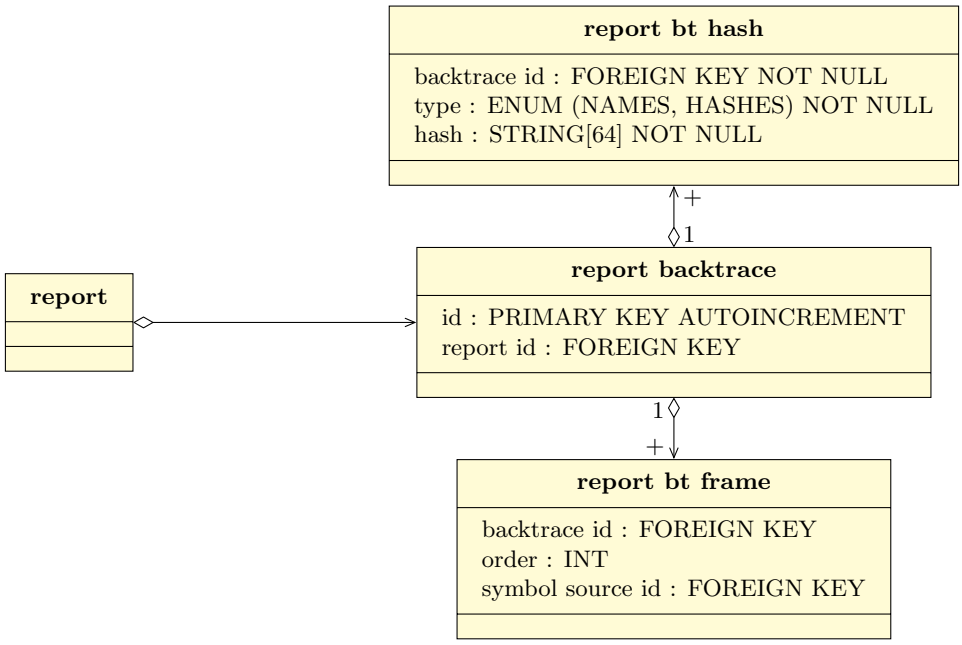
Problems are dynamically created from reports.

5.3.2 Problem Storage – Reports

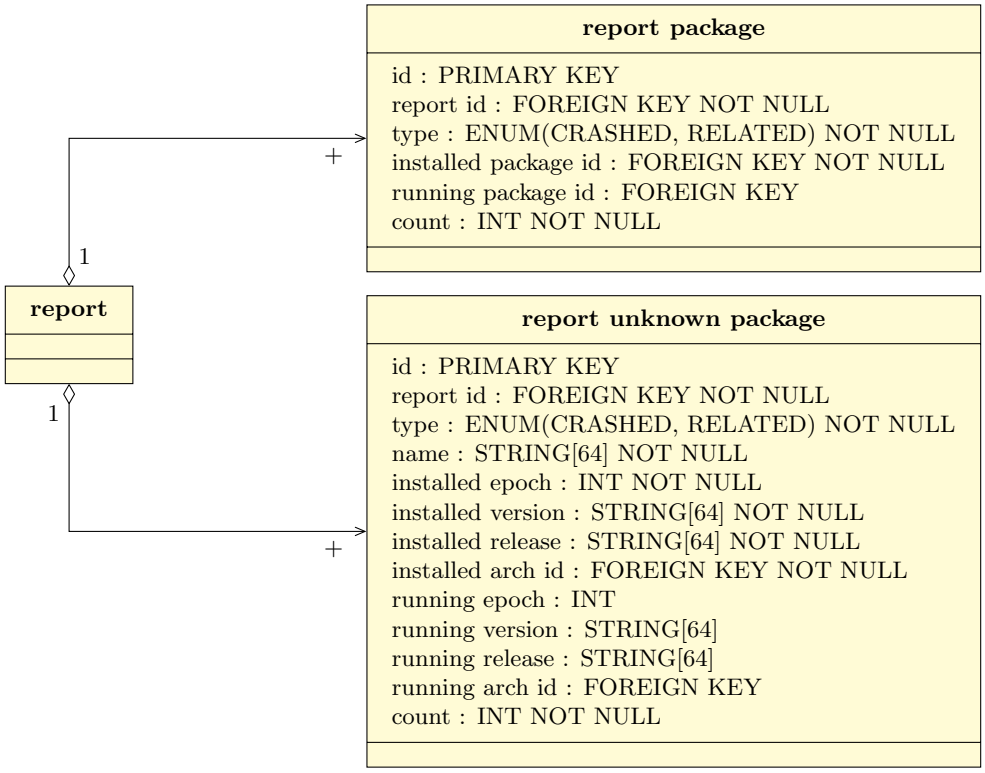
Part of reports is populated from μreports.



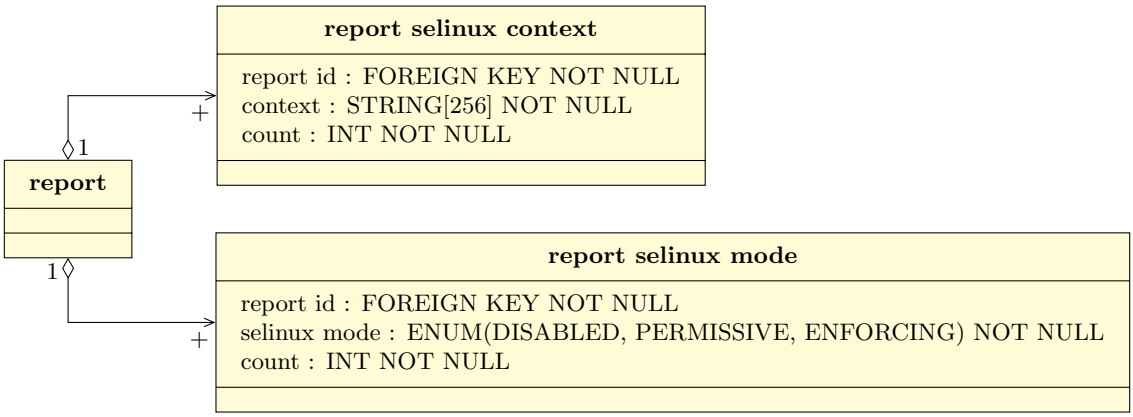
5.3.3 Problem Storage – Report backtraces



5.3.4 Problem Storage – Report packages

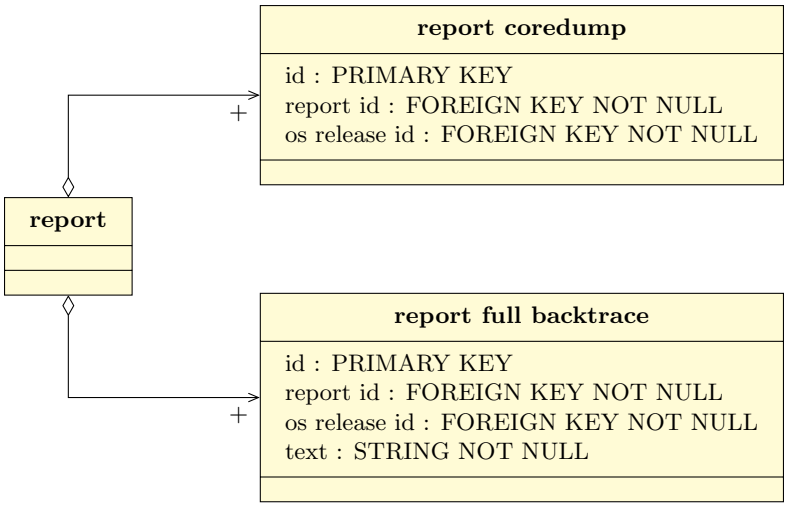


5.3.5 Problem Storage – Report security aspects

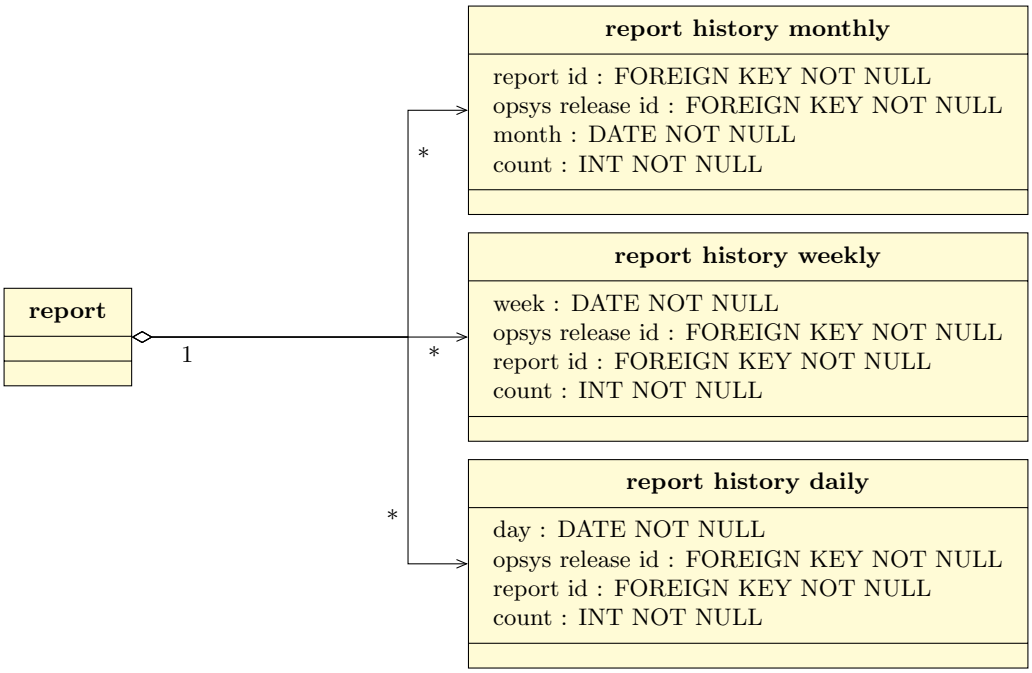


5.3.6 Problem Storage – Report in-depth information

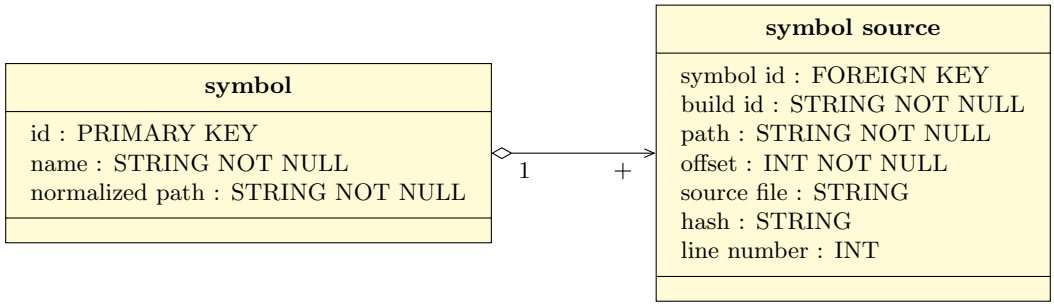
Part of reports is populated from full reports.



5.3.7 Problem Storage – Report history

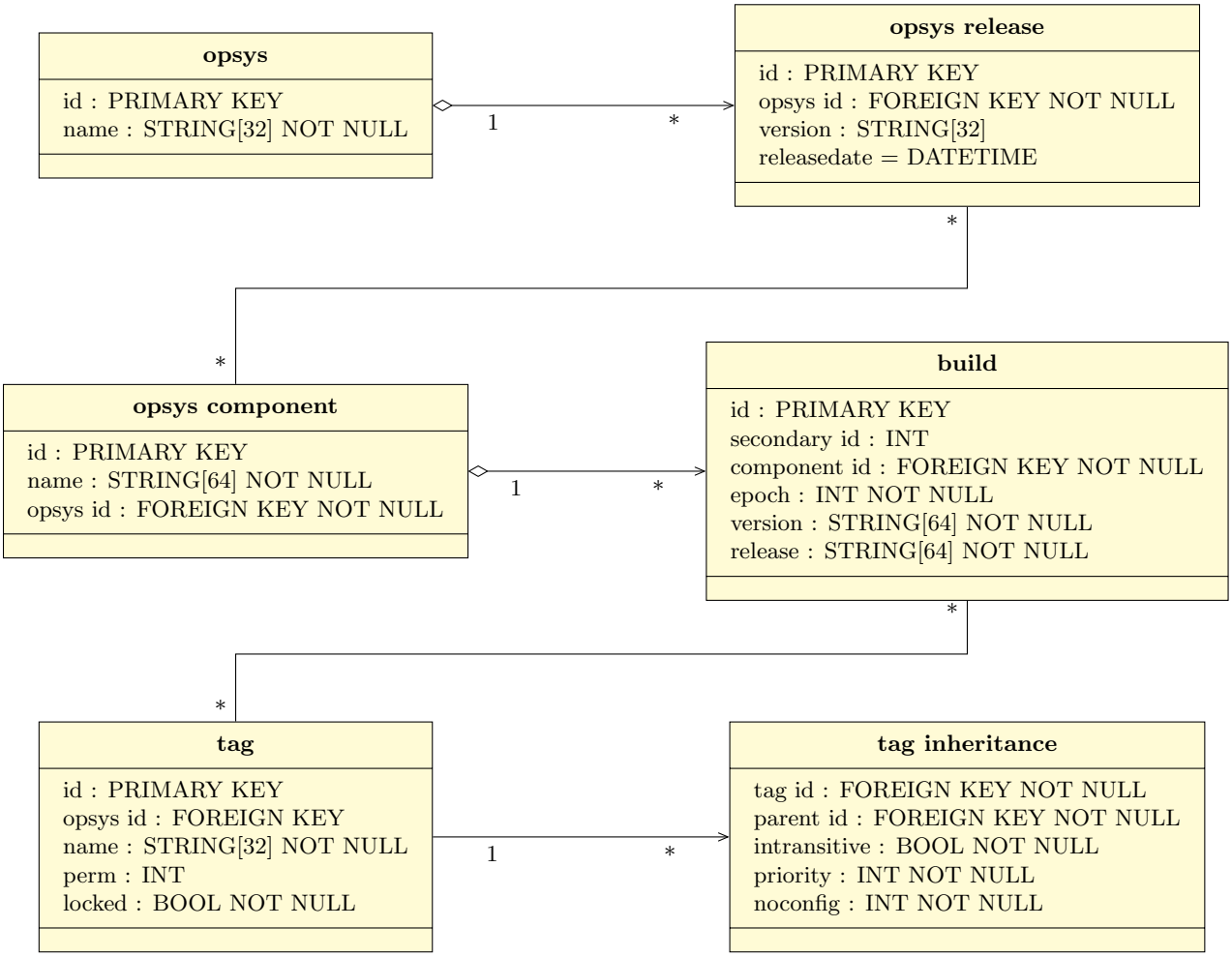


5.3.8 Problem Storage – Symbols

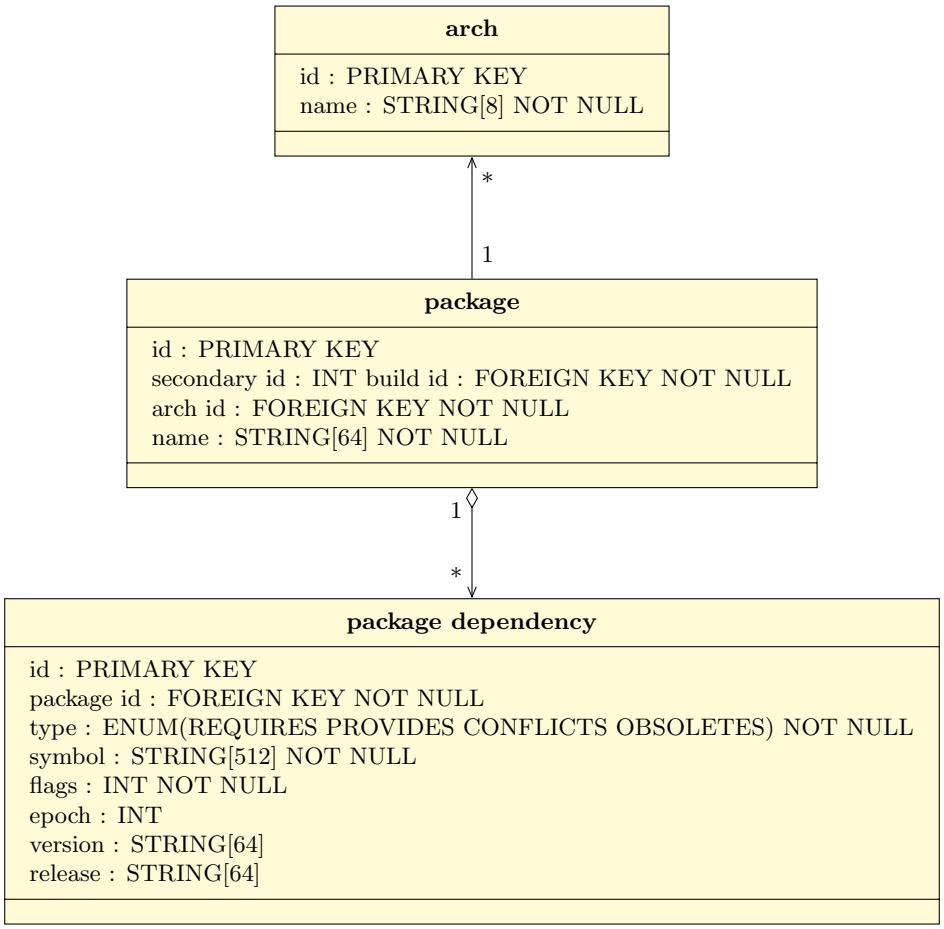


5.3.9 Problem Storage – Clusters

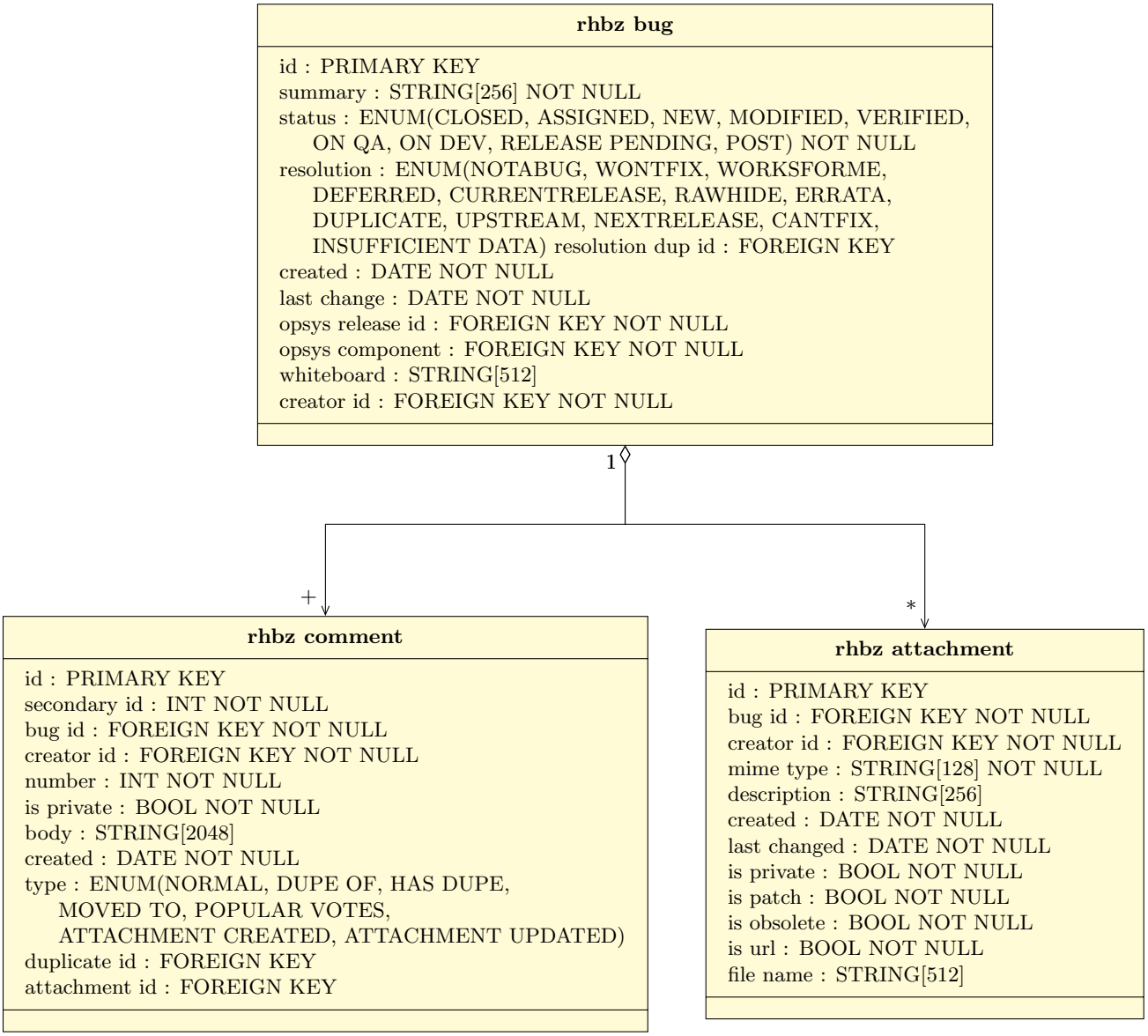
5.3.10 Data Storage – Builds



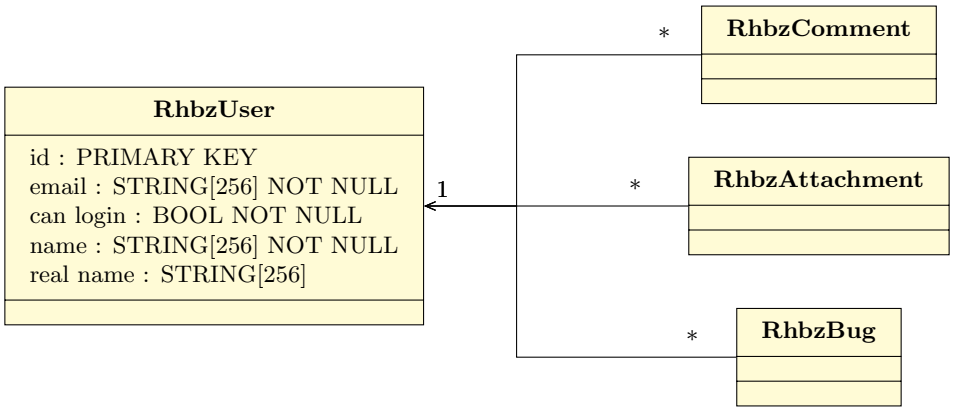
5.3.11 Data Storage – Packages



5.3.12 Data Storage – Red Hat Bugzilla



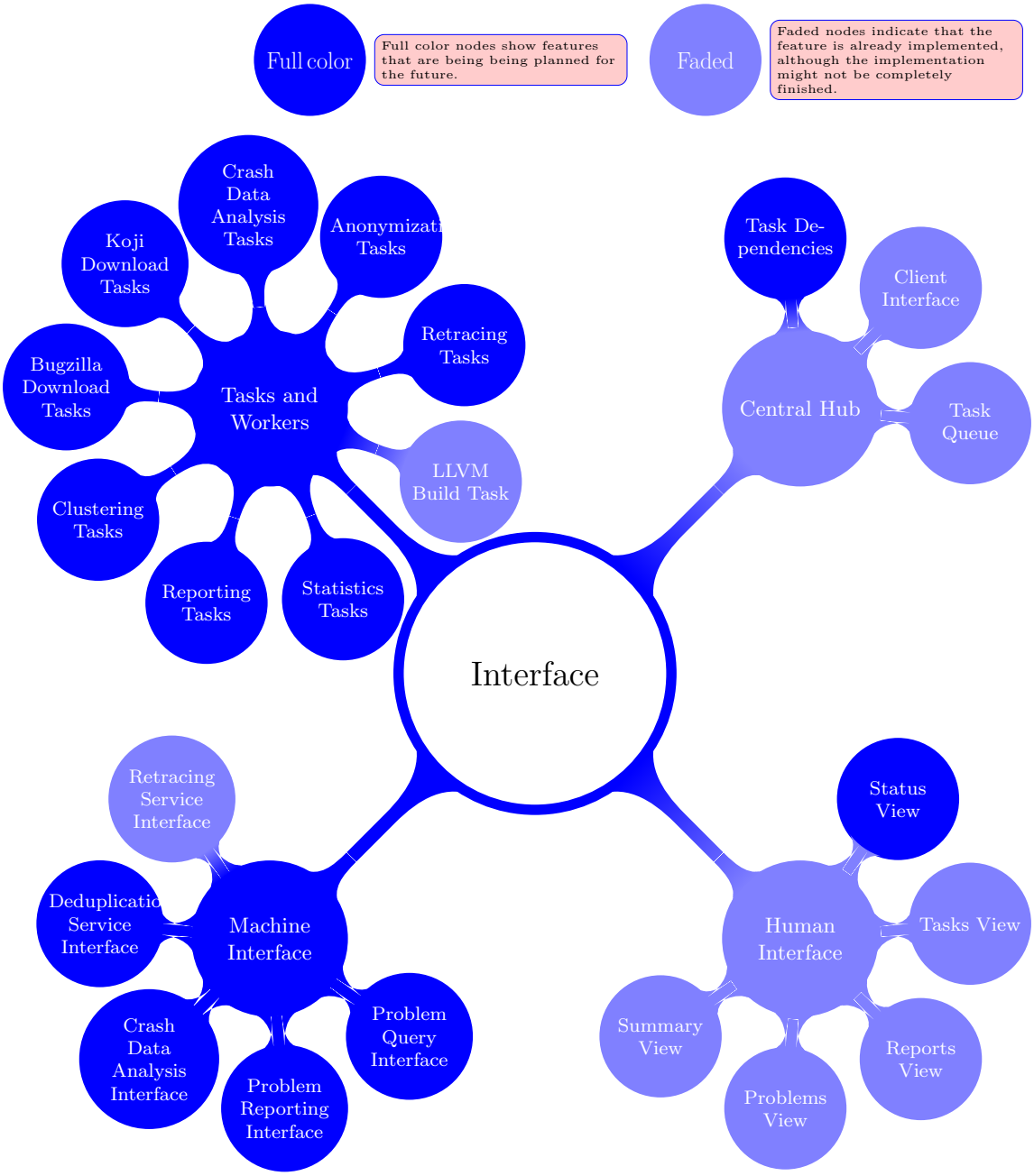
5.3.13 Data Storage – Red Hat Bugzilla Users



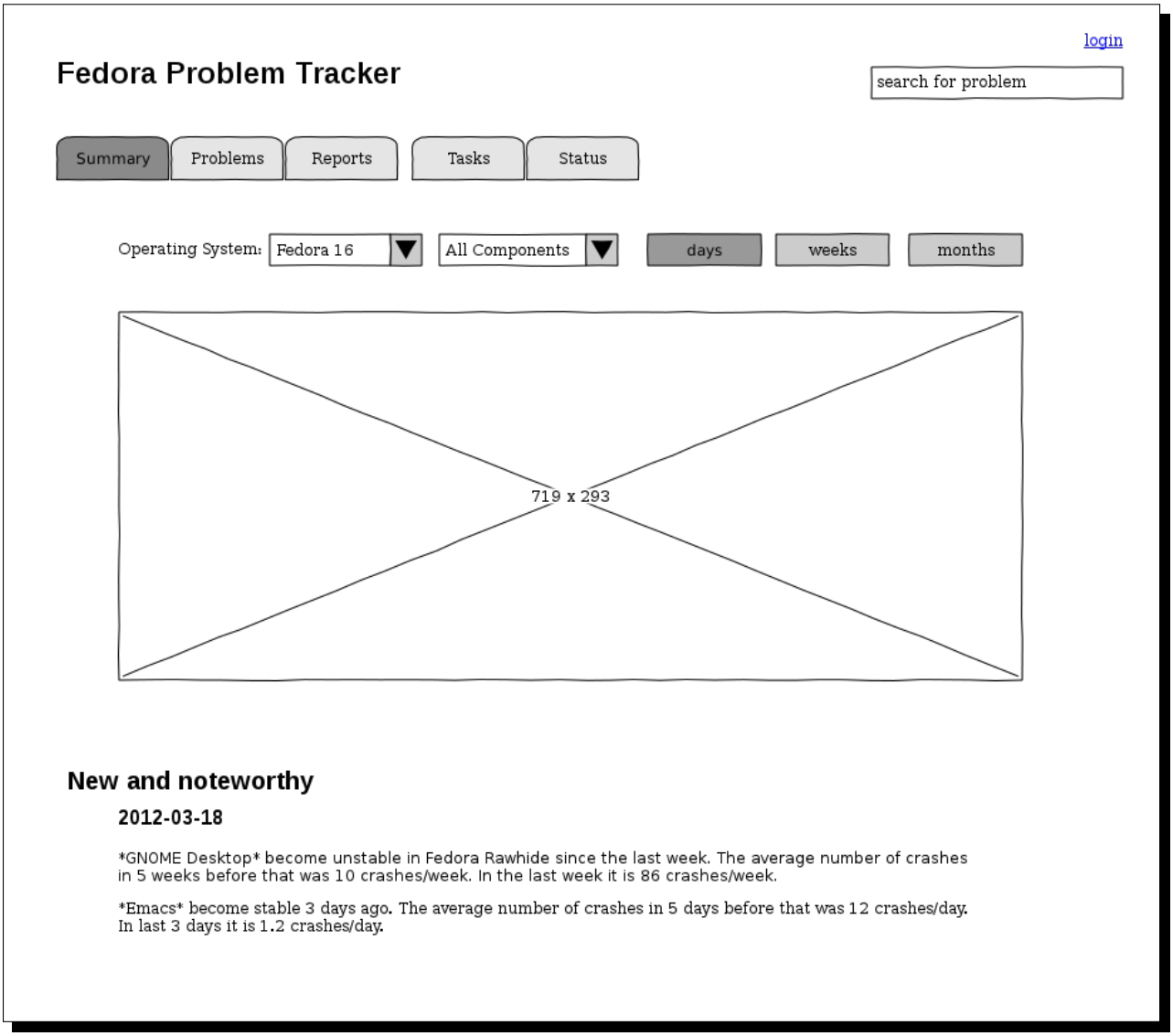
5.3.14 Data Storage – LLVM Bitcode

5.3.15 Sanity Checker – Debuginfo checker

5.4 ABRT Server Interface Overview



5.4.1 Summary Page



Operating System The available options include Fedora releases, pre-release branched Fedora, Fedora Rawhide, All. The list of releases should be obtained from Fedora Package Database (this is handled by Storage Synchronization.

Components The available options include All Components, the list of all components for the selected Operating System, and selected `comps.xml` groups (such as GNOME Desktop, KDE Desktop, Xfce, Web Server, Electronic Lab, Engineering and Scientific, Font design and packaging, System Tools, Sound and Video, Office/Productivity)

Graph Displays the number of problems (individual events) in time. The underlying data are updated once a day.

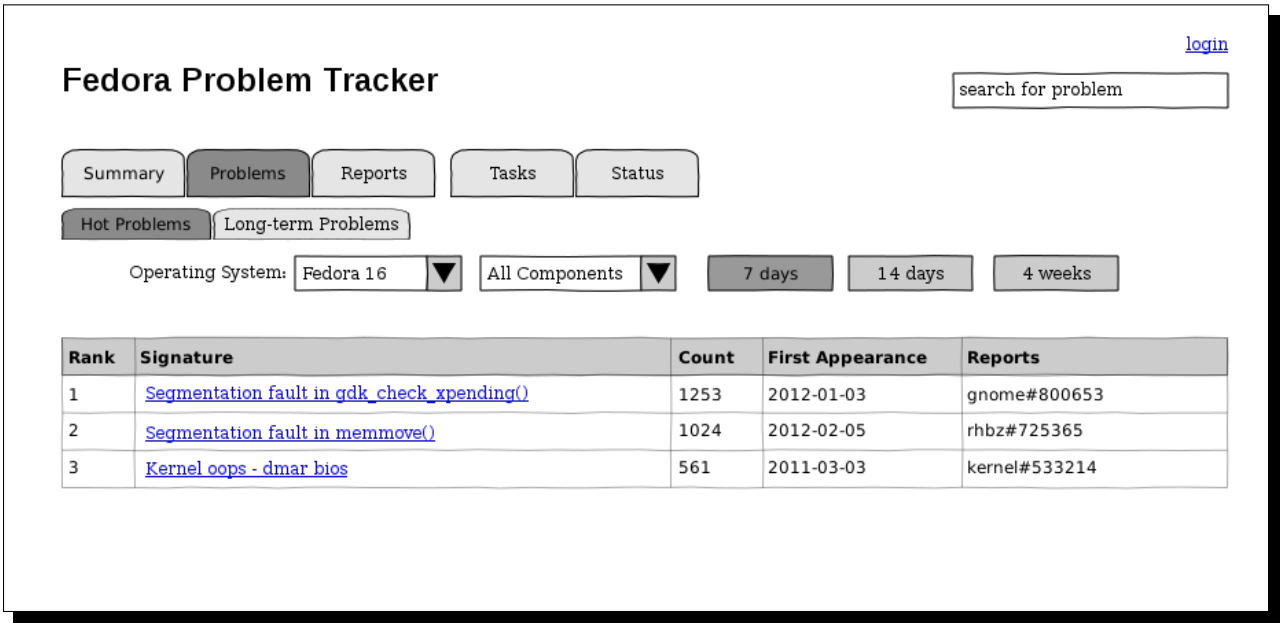
Days The graph shows problems in the last 14 days.

Weeks The graph shows problems in the last 12 weeks (3 months).

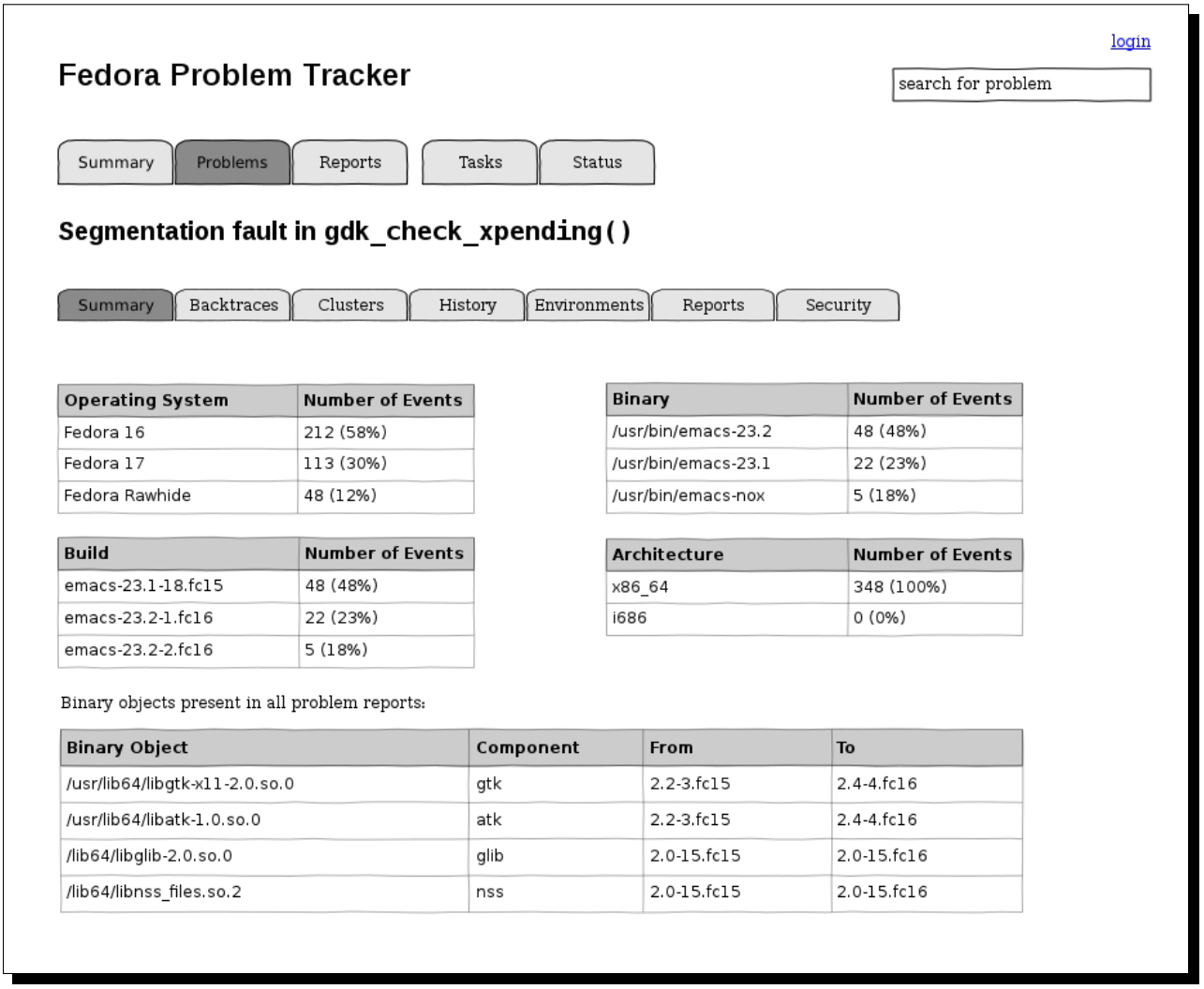
Months The graph shows problems in the last 12 months.

New and noteworthy Shows automatically discovered interesting trends that can be detected from data. It tells visitor which combinations of Operating System and Component might be worth looking.

5.4.2 Problems Overview Page



5.4.3 Problems Item Summary Page



login

Fedora Problem Tracker

search for problem

Summary | Problems | Reports | Tasks | Status

Segmentation fault in gdk_check_xpending()

Summary | Backtraces | Clusters | History | Environments | Reports | Security

Operating System	Number of Events
Fedora 16	212 (58%)
Fedora 17	113 (30%)
Fedora Rawhide	48 (12%)

Binary	Number of Events
/usr/bin/emacs-23.2	48 (48%)
/usr/bin/emacs-23.1	22 (23%)
/usr/bin/emacs-nox	5 (18%)

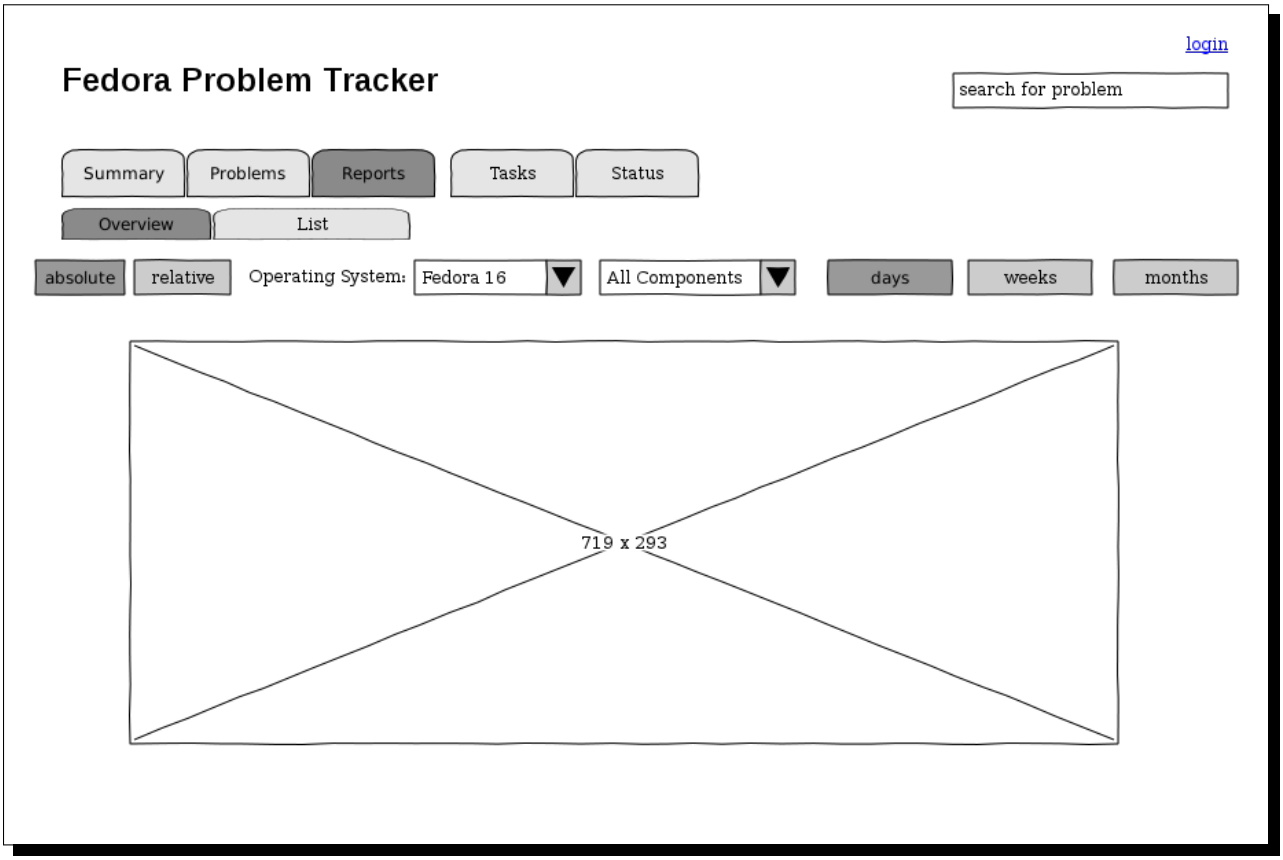
Build	Number of Events
emacs-23.1-18.fc15	48 (48%)
emacs-23.2-1.fc16	22 (23%)
emacs-23.2-2.fc16	5 (18%)

Architecture	Number of Events
x86_64	348 (100%)
i686	0 (0%)

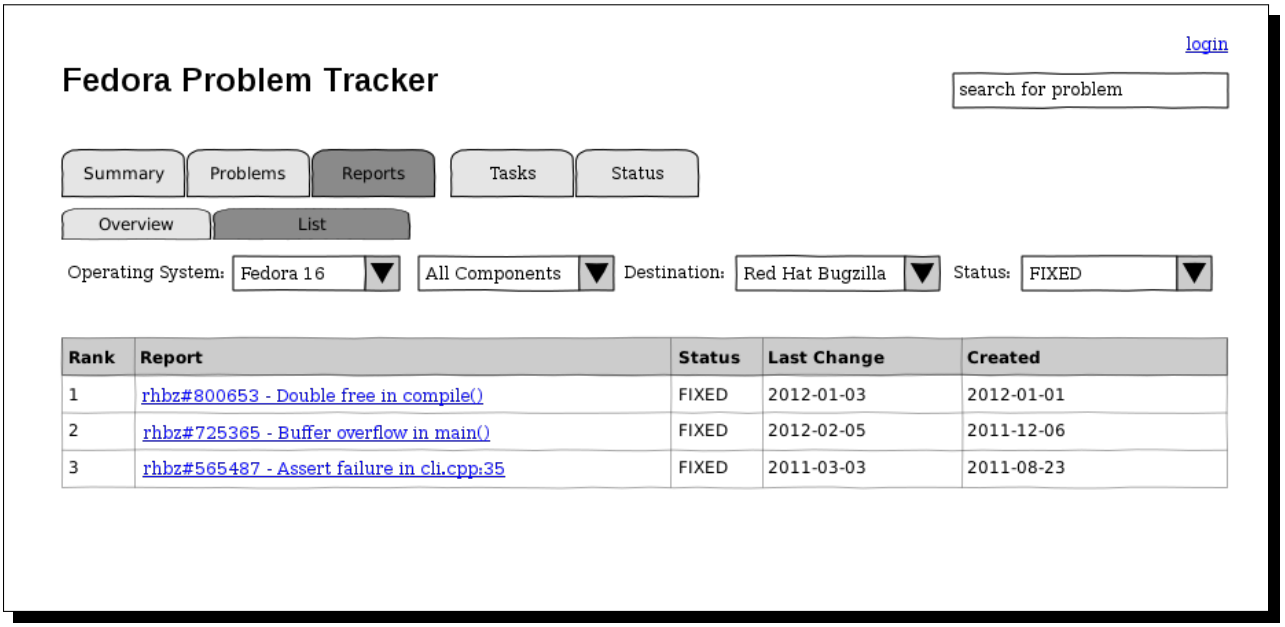
Binary objects present in all problem reports:

Binary Object	Component	From	To
/usr/lib64/libgtk-x11-2.0.so.0	gtk	2.2-3.fc15	2.4-4.fc16
/usr/lib64/libatk-1.0.so.0	atk	2.2-3.fc15	2.4-4.fc16
/lib64/libglib-2.0.so.0	glib	2.0-15.fc15	2.0-15.fc16
/lib64/libnss_files.so.2	nss	2.0-15.fc15	2.0-15.fc16

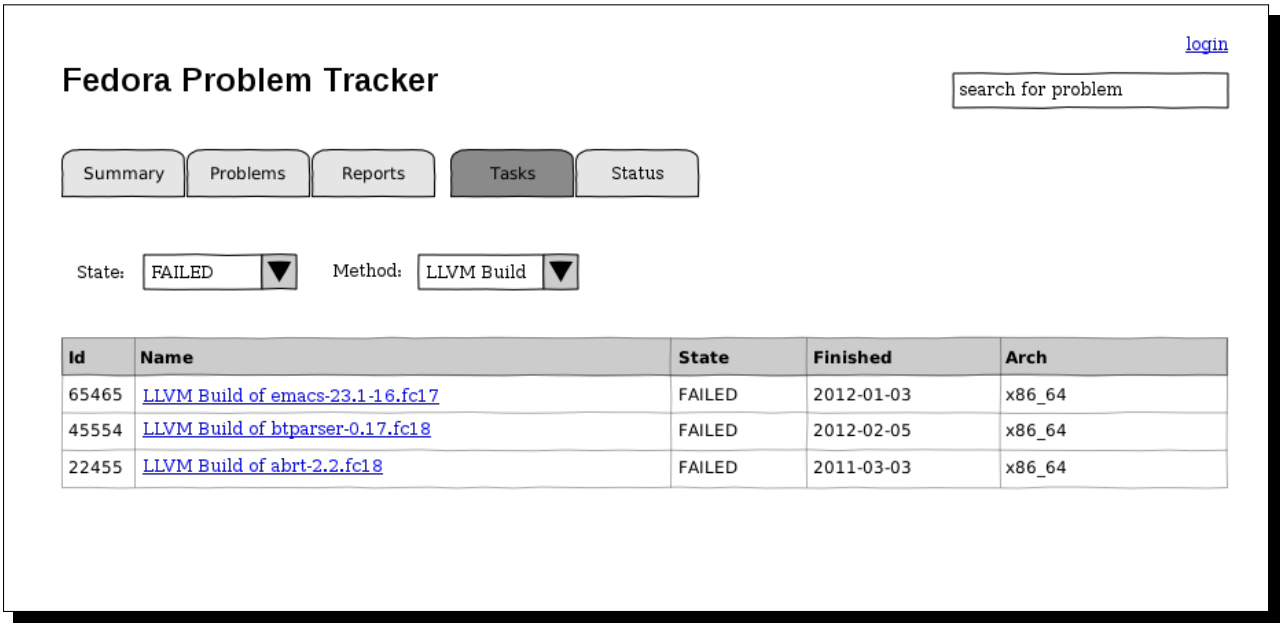
5.4.4 Reports Overview Page



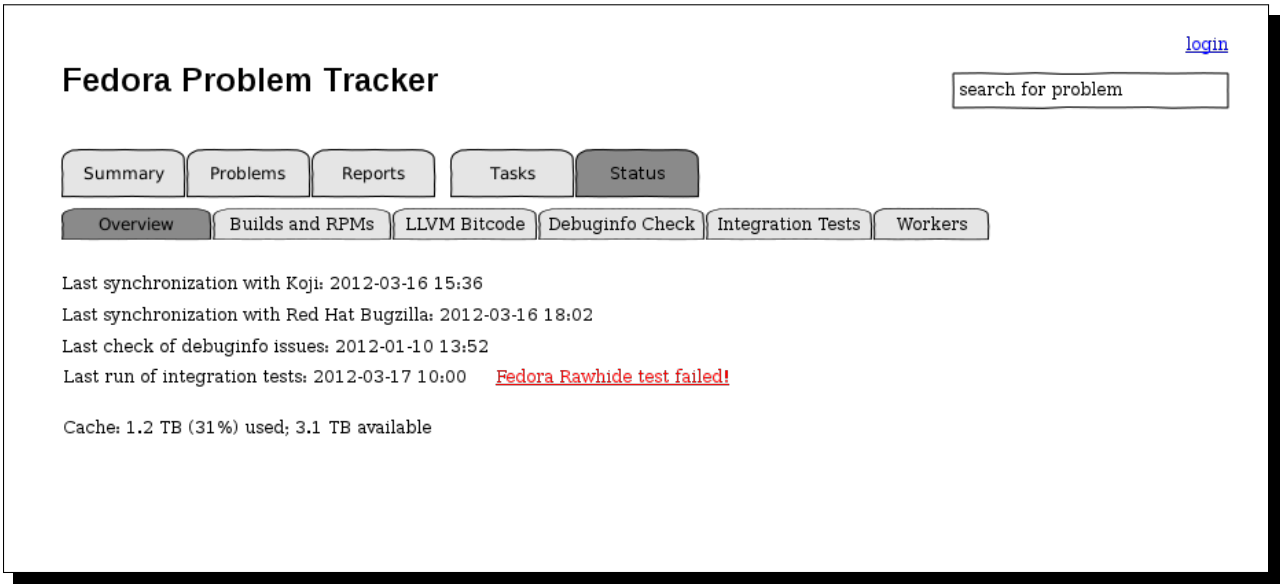
5.4.5 Reports List Page



5.4.6 Server Tasks Page



5.4.7 Server Status Page



5.4.8 Problem Reporting Interface

Server accepts microreports. Microreport is a JSON-formatted data structure described below:

Name	Format	Mandatory	Notes
`type`	`python`, `userspace`, or `kerneloops`	yes	
`reason`	Unicode string, max. 128 characters.	yes	Format depends on the `type`.
`uptime`	Unsigned integer.	yes	Number of seconds from program start to the problem.
`executable`	Full path, max. 512 characters.	yes	
`installed_package`	Dictionary; see the `package` table below.	yes	
`running_package`	Dictionary; see the `package` table below.	no	
`related_packages`	List of dictionaries; see the `related_package` description below.	yes	
`os`	Dictionary; see the `os` table below.	yes	
`architecture`	`x86_64` or `i386`	yes	
`reporter`	Dictionary; see the `reporter` table below.	yes	Program that created the report.
`core_backtrace`		yes	
`os_state`	Dictionary; see the corresponding table below.	yes	
`user_type`	`root`, `nologin`, `local`, or `remote`	no	
`selinux`	Dictionary; see the corresponding table below.	no	SELinux presence and mode.
`proc_status`	ASCII string, max. 2 kB.		The contents of `/proc/pid/status`.

The `os` structure:

Name	Format	Mandatory	Notes
`name`	ASCII string.	yes	
`version`	ASCII string	yes	Numeric. No codenames.

The `os_state` structure:

Name	Format	Mandatory	Notes
`suspend`	`yes` or `no`	no	Problem happened during suspend, hibernate, or resume.
`boot`	`yes` or `no`	no	Problem happened during boot process.
`login`	`yes` or `no`	no	Problem happened during login process.
`logout`	`yes` or `no`	no	Problem happened during logout process.
`shutdown`	`yes` or `no`	no	Problem happened during the shutdown process.

The `reporter` structure:

Name	Format	Mandatory	Notes
`name`	ASCII string, max. 128 characters.	yes	
`version`	ASCII string, max. 128 characters.	yes	

The `related_package` structure:

Name	Format	Mandatory	Notes
`installed_package`	Dictionary; see the `package` table below.	yes	
`running_package`	Dictionary; see the `package` table below.	no	

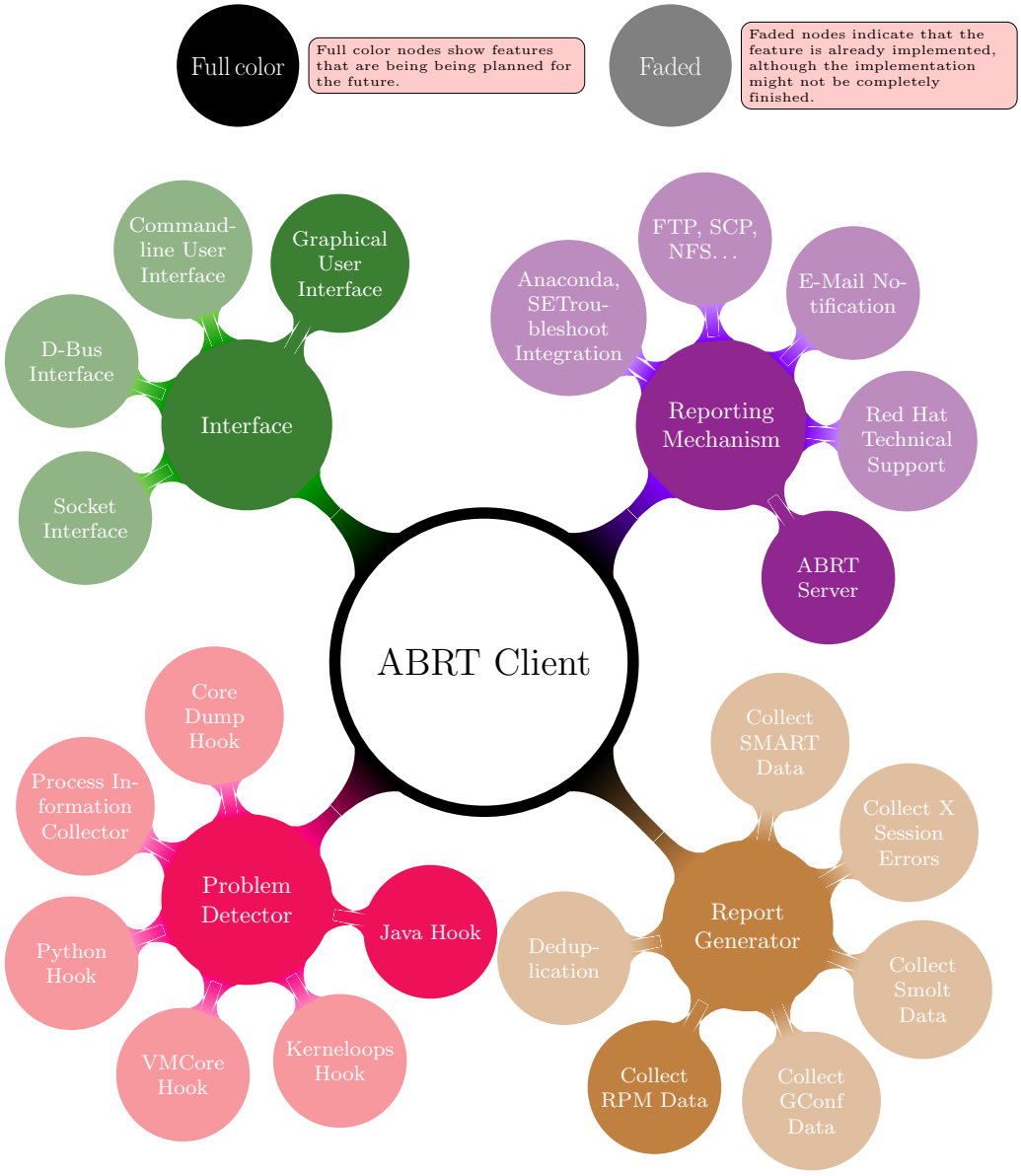
The `package` structure:

Name	Format	Mandatory	Notes
`name`	ASCII string, max. 128 characters.	yes	
`version`	ASCII string, max. 128 characters.	yes	
`release`	ASCII string, max. 128 characters.	yes	
`architecture`	ASCII string, max. 128 characters.	yes	
`epoch`	ASCII string, max. 128 characters.	yes	

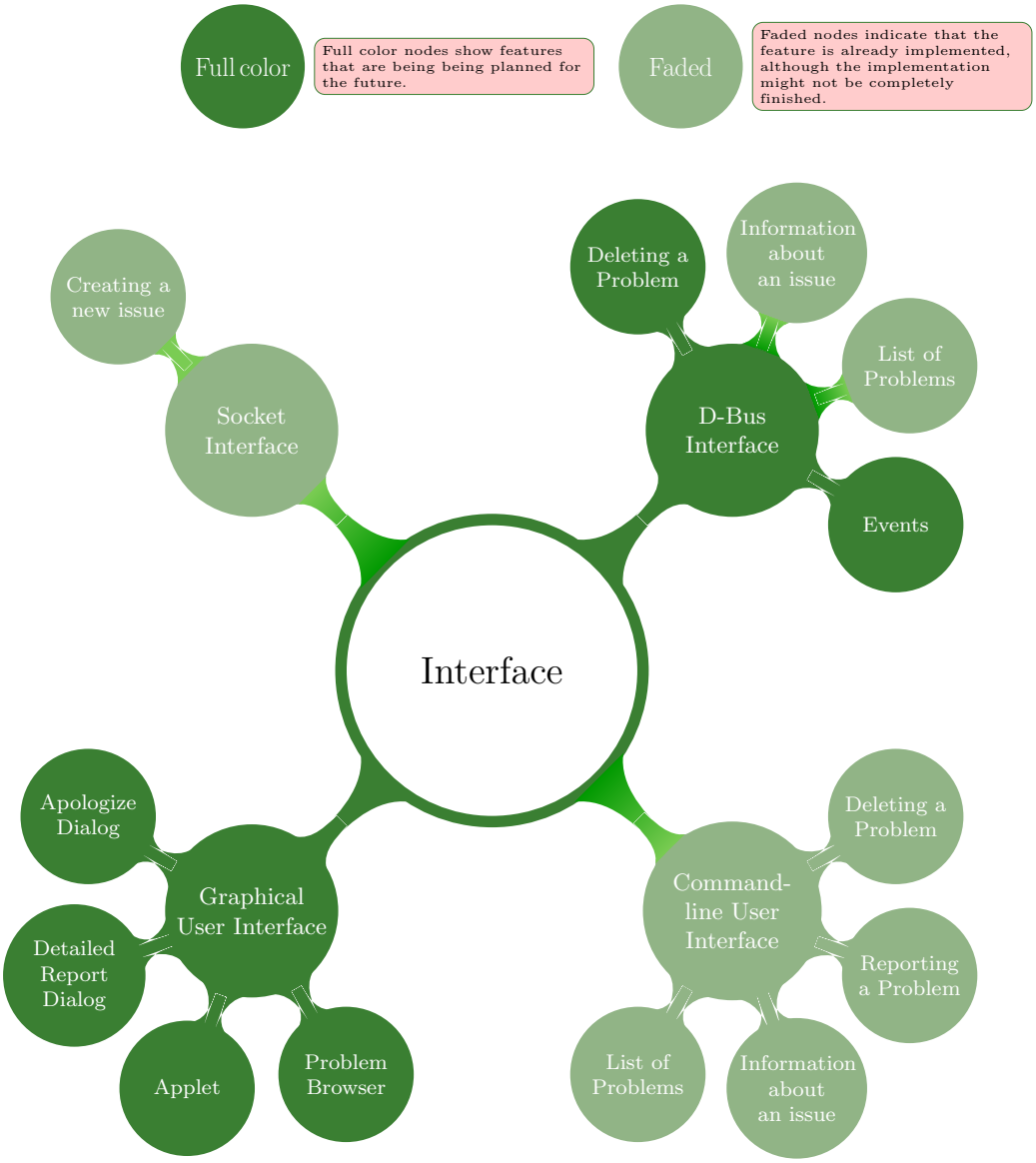
The `selinux` structure:

Name	Format	Mandatory	Notes
`mode`	`enforcing`, `permissive`, or `disabled`	yes	
`context`	ASCII string, max. 128 characters	yes, if the `mode` is either `enforcing` or `permissive`	`ps -e --context`
`policy_package`	Dictionary; see the `package` description.	no	

5.5 ABRT Client Overview

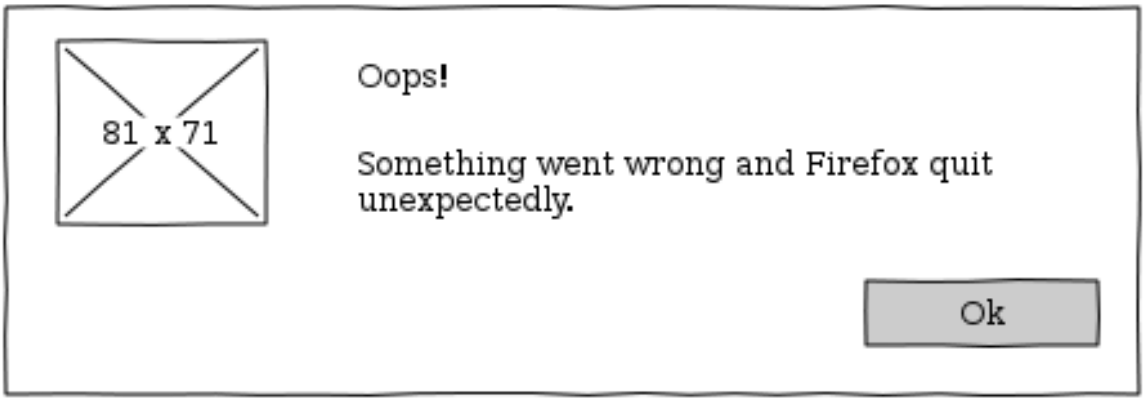


5.6 ABRT Client Interface Overview



5.6.1 Apology Dialog

The dialogs come from [1]. Please see [1] for more detailed design.

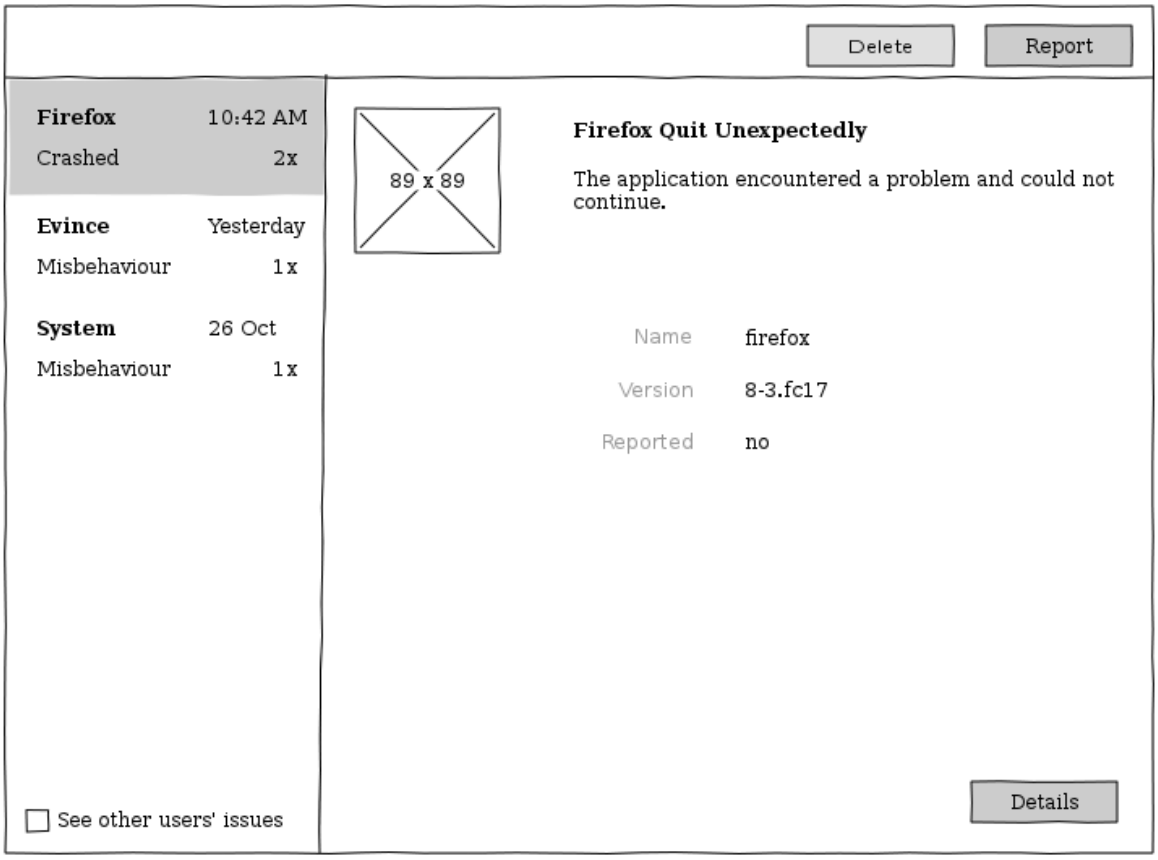


5.6.2 Detailed Report Dialog

5.6.3 Applet

5.6.4 Problem Browser

The Problem Browser dialog comes from [1]. Please see [1] for more detailed design.



5.7 ABRT Client Reporting Mechanism Overview

5.7.1 Reporting to ABRT Server

5.8 ABRT Client Report Generator Overview

5.8.1 Coredump-level backtraces

6 Project Time Management

6.1 Fedora 17 Phase

Finish date: approx 2012-06-01.
Consists of 3 sprints, 3 weeks each.

The goals for the ABRT Server to be reached at the end of the phase:

Usable human user interface Server has initial but usable graphical user interface. It shows problems, reports, and report history in a graph.

μReport processing Server accepts μreports, stores them to the database, retraces the contained symbols. Accepted reports are visible in the user interface.

Clustering of reports into problems Report clusters (problems) on server are generated using core-backtrace distance.

The goals for the ABRT Client to be reached at the end of the phase:

μReport sending Client sends μreports from the crash dialog.

6.1.1 Sprint 1

Finish date: 2012-03-20 Tuesday
Duration: 3 weeks
Status: FINISHED

- Server
  - Problem storage
    - Database schema [mtoman,mlichvar]
    - Storage of incoming problems [mlichvar]
  - Data storage
  - Initial database schema [mtoman]
  - Deduplication
    - Deduplication of incoming problems according to hashes
    - Deduplication of retraced problems according to symbols
  - Retracing
    - Retracing of microreports
  - Machine interface
    - Receive report
  - Migration to SQLAlchemy [mlichvar]
  - RHEL6 compatibility [mtoman]
- Client
  - Coredump-level backtraces [mmilata]
  - Microreport sender [npajkovs]
  - User interface [dvlasenk]

6.1.2 Sprint 2

Start date: 2012-04-02 Monday
Finish date: 2012-04-27 Friday
Duration: approx. 3 weeks

- Server
  - Human interface
  - Machine interface
  - Tasks and workers
  - Clustering
    * Adapt existing source code to match the server
    * Properly create problems from reports

6.1.3 Sprint 3

Start date: 2012-05-07 Monday
Finish date: approx. 2012-06-01 Friday
Duration: approx. 4 weeks

- Accepting reports on the server
- Python and Koops μreports
- Rewrite cache to storage
- Generating problems
- Deploy server internally

6.2 Fedora 18 Phase

Finish date: approx. 2012-11-01

The goals for the ABRT Server to be reached at the end of the phase:

Retrace server merged Retrace server functionality is merged into the server, using server's packages, server's *chroot* implementation and server's *libsolv* integration. The original retrace server is still maintained to support current deployments.

Debuginfo check Server checks the consistency of packages containing the debugging symbols for program binaries. It supports opening bugs for packages with broken debugging symbols.

LLVM support Server handles LLVM rebuilds of packages to allow static analysis of source code and source code browser.

Full reports Server requests and stores full report (coredump) for problems that require fixing due to high priority.

Better human interface Problems and reports can be studied by users in a good detail. Server shows its status (which packages are supported, internal task failures such as failed package download or LLVM rebuild).

Machine interface Problems and reports can be fetched via JSON. Problems, reports, and Red Hat Bugzilla bugs can be searched by providing a backtrace. Server can be used to obtain the crash function from a textual backtrace.

The goals for the ABRT Client to be reached at the end of the phase:

Better human interface

6.2.1 Sprint 4

Start date: approx. Duration: approx. 3 weeks

- Determine crash function from core backtrace
- Problem names and report names
- Finish retracing
- JSON interface for search with a backtrace
- LLVM rebuild
- Builds and packages page in Status
- Overview page in Status
- RHBZ Bugs in Status

6.2.2 Sprint 5

Duration: approx. 3 weeks

- Extend LLVM rebuild with DXR for source code browsing.
- Interface for source code browsing (clickable function names in coredump-level backtraces.
- Request and store coredumps and vmcores.
- Implement coredump and vmcore retracing.
- JSON API for coredump and vmcore retracing.

6.2.3 Sprint 6

Duration: approx. 3 weeks

- LLVM linking into single module
- Component groups
- Debuginfo checker

6.2.4 Sprint 7

Duration: approx. 3 weeks

- Problem type analysis start
- Security analysis start
- Merge All/Running/Finished Tasks in Status into single page.

6.2.5 Sprint 8

Duration: approx. 3 weeks

- Bugfixing
- Fedora deployment

6.3 Fedora 19 Phase

The goals for the ABRT Server to be reached at the end of the phase:

Automatic reporting to Red Hat Bugzilla

Security analysis of reports

Static analysis of SELinux AVCs

6.3.1 Sprint 9

- SELinux AVCs
- Reporting to Red Hat Bugzilla

6.3.2 Sprint 10 and later

- Reporting to upstreams
- SELinux analysis
- Symbolic execution
- Crash analysis

7 Related Work

There are several existing implementations of software failure management systems.

7.1 GNOME Problem Reporting

Jon McCann of GNOME team envisioned a problem reporting architecture[3] that splits the responsibilities of a problem management system into several components:

1. *System Logger* collects data for anomalous behavior of system such as crash dumps and SELinux access denial logs. It is proposed to include this functionality into `systemd`, a system and service manager for Linux. In internal communication, Jon also proposed to stop using core dumps in favor of minidumps.

2. *Problem Detector* watches the output of System Logger for new events, and runs Report Generator on every new event. Its name should be `problemd`, and this tool is not implemented yet.

3. *Report Generator* gathers supplementary details about a problem, and stores problem data to a non-volatile memory. It should be handled by either `systemd` or `problemd`.

4. *User Problem Notifier* notifies user about a problem. Jon proposes to include this functionality into `gnome-settings-daemon`.

5. *Reporting Mechanism* delivers problem report to a Collection Server, scp, ftp, email...

6. *Problem Reporting and Review* shows problem reports of a system, and allows report submission. Jon designed *Oops!*[1], a graphical user interface for such a component.

7. *Problem Report Collection Server* accepts anonymous submissions, supports filling reports to Bugzilla, scrubs sensitive data from reports, detects duplicates, performs coredump analysis and retracing (generating backtrace from coredump). In internal communication, GNOME team proposes using Socorro, a crash statistics server project of Mozilla, in this role.

Advantages and good aspects of the proposal.

1. The idea of pushing generic code to packages that are being used across distributions. Coredump catching can definitely be done by `systemd`. *Problem Detector* and *Report Generator* can also be made portable and shareable across distributions, and it could live in `http://freedesktop.org` as `problemd`.

2. The design of *Oops!*.

Criticism of the proposal. The most important point that should be re-evaluated is the proposed direction of *statistics-based* problem management and bugfixing. This includes the usage of minidumps in all situations, and the deployment of Socorro.

Data from the current problem management system shows that statistics-based bugfixing misfits the operating system level use case (management of full stack of applications). Statistics-based bugfixing is based on the assumption of large amounts of users hitting and reporting the same crash. This assumption is valid for desktop applications, which are quickly changing, contain large amount of bugs (GUI code is difficult to handle well), and have large amount of users (desktop shell, internet browser, e-mail client). Nevertheless, this assumption is *invalid* for most of operating system packages, including server-side software. Many packages are used by relatively low number of people, customized and deployed just on a few servers. Crashes in this situation are less frequent, but dangerous.

1. Implementing *User Problem Notifier* into `gnome-settings-daemon` brings the requirement to implement the same functionality for other desktops as well (KDE, XFCE).

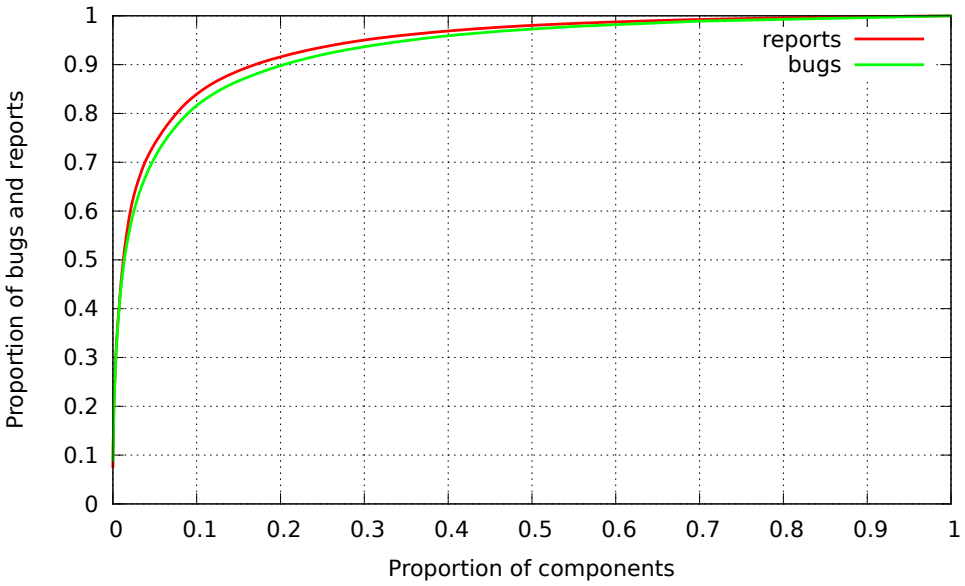


Figure 1: Cumulative distribution of ABRT bugs and reports per component

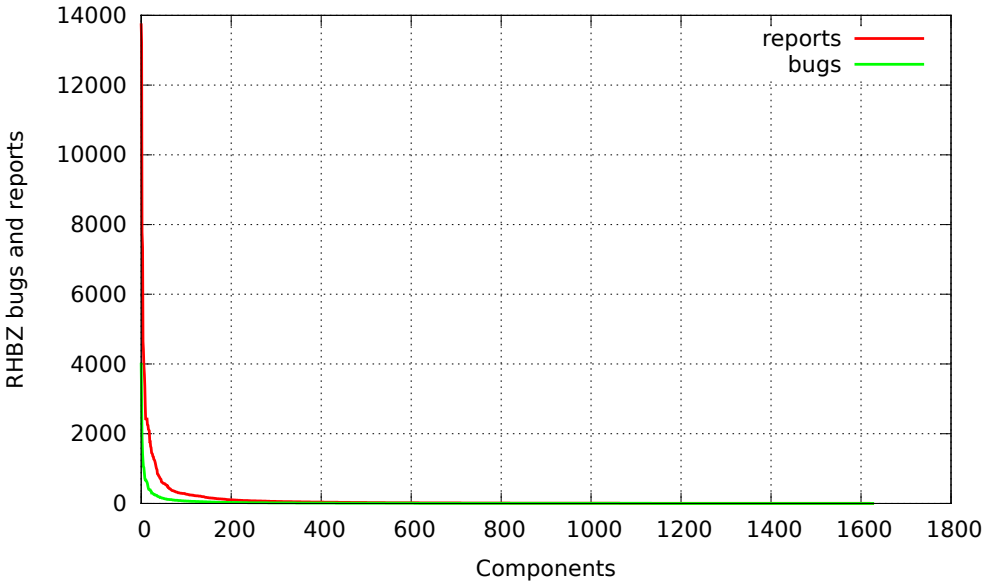


Figure 2: Distribution of ABRT bugs and reports per component

2. *Oops!* design is incomplete. It is not clear how the reporting target can be configured (problem report collection server URL, e-mail, FTP, SCP destinations). The window with report details is not presented.

7.2 Windows Error Reporting

TODO: [4].

7.3 Ubuntu Apport

References

[1] Jon McCann, *Oops!*. https://live.gnome.org/Design/Apps/Oops.

[2] Jon McCann, *Problem Recovery and Reporting*. https://live.gnome.org/GnomeOS/Design/Whiteboards/ProblemReporting.

[3] Jon McCann, *Problem Reporting Architecture Proposal*. https://live.gnome.org/GnomeOS/Design/Whiteboards/ProblemReporting/Proposal.

[4] *Debugging in the (Very) Large: Ten Years of Implementation and Experience*. http://msdn.microsoft.com/en-us/windows/hardware/gg487440.