Interface Stability and System Evolution

Edward Hunter, Joe Kowalski, John Plocher

Abstract: Internal to Sun Microsystems we have a process we follow to track and document the stability of the interfaces used by the system as a way of maintaining compatibility. This paper will describe that process and some of the things we have learned about interfaces and controlling them in a large complex system (Solaris) being worked on by multiple developers in many different locations.

The Problem we tried to solve

In the early days of Sun Microsystems (the early 90's) the company was faced with a variety of challenges around evolving our system software to meet the needs of our customer base. Sun had requirements for changes to SunOS (and later Solaris) to add new functionality and maintain the investment in software our customer base had made. We viewed this investment as the ability to continue to run customer applications with no or little change as the operating system platform evolved.

Sun chose two fundamental premises around which to build a set of policies and procedures for doing development inside of the company.

1) Find ways to accelerate the rate of adoption of change in our customer base.
2) Find ways to reduce the customer effort to test and validate their software applications against our platform.

From these precepts eventually sprung the Solaris Application Binary Guarantee. The guarantee says in effect that between releases of Solaris, applications which use the interfaces (API's) we consider to be stable will continue to work without change to the application. We chose initially to focus on the interfaces between the operating system and the user applications. These were essentially a contract between our customers and us. As such we are careful about changes to then that might affect the customer applications.

In those days the thing Sun strove for was to attract ISV's to our platform. To do this we had to focus on at least three things: performance, ease of porting and stability. This paper will not deal with the performance aspect; instead it will focus on things that facilitated porting and stability of the ISV applications. On the porting and stability side we asserted that getting the ISV's to focus on a set of interfaces rather than system behaviors would be better for all in the long run. It would give us the freedom to innovate behind the interfaces and it would give the ISV's fewer things to pay attention to.

To try and give the ISV's (and ultimately customers) a better sense for what to expect with any particular release of a product we adopted a naming scheme of major/minor/micro releases. Each type of release allowed a more restrictive set of changes to interfaces to be made relative to their stability with the previous release. For instance, major releases allowed for incompatible changes to be introduced in the interfaces, while micro releases were pretty much only bug fixes (no new functionality initially). We should point out that a product can contain other products that may have their own release versions.

This paper will describe some of the systems we created (in particular the architecture review process) to help achieve the two goals stated above.

Some Early History

Historically, Sun has relied on many small independent engineering groups to develop different parts of the system. We strove to create an environment where these groups could create their projects independently and integrate them together with projects from other parts of the organization to form the whole product.

Note that this approach is similar to what's found in the free software community and other organizations. The main difference being some centralized coordination driven by management to achieve certain

strategic and tactical goals. In particular since we also shipped hardware there was a need to align the efforts of the hardware groups and the software groups.

For this to work, a project team had to know a couple of things outside of the sphere of their project. First, who controlled the other parts of the system that they were not working on. Second, which parts of the surrounding system were available for their use. More directly, they needed to understand who owned a particular piece of code and which interfaces to that code were appropriate for consumption by their project.

As Sun grew we discovered that it was hard to keep track of both of these things. This lead to situations where projects would depend on an interface that wasn't really appropriate only to discover late in the game that it had changed, causing redesign and rework on their part. In one particular case we had two projects delivering different versions of the same header file for customers to use. This set up the situation where both products could not be installed on the system at the same time. Problems like that lead us to examine the process for development with an eye towards reducing these types of errors.

Our solution

One thing we decided early on is that having a record of the decisions we made would be very important. Effectively having a database that described the interfaces in the system and how the different pieces connected together was the goal. Additionally, having some written record of why certain decisions were made and what the inputs were to those decisions was considered important. One side benefit of doing things this way was to centralize all the inputs that went into making a decision. This centralization makes it easy for people coming along after the fact to track down details of a case without trying to find the particular email thread on an alias where the issues came up. Part of the infrastructure maintains a case log of all current and past cases, which is easily searchable. This log points to the case directory for each case that contains all the materials including the email discussion.

Conceptually, the process we developed has some similarities to the U.S. court system without the high priced attorneys. Projects that are to be reviewed are referred to as cases. Opinions are written to capture the rationale behind the approval (or denial) of the case and suggest areas for improvement. If there are issues with the decision it may be appealed to a higher level.

We chose to have several committees called Architecture Review Committees (ARC) to review the cases. Think of these as judges who volunteer from the developer community to provide expertise on different subject areas. It should be pointed out that the ARC members are all experienced senior engineers with domain expertise in different parts of the system.

Each case maintains a case history that includes all the documents that describe the case, meeting minutes and the email threads where discussions happen. Capturing the rationale enhances the consistency of decisions and the ability to reevaluate decisions as the computing environment evolves.

We chose to initially focus on the binary interfaces (the ABI) and later expanded that to additional things that might be consumed programmatically (see the definition of interface in next section). This would include file system location, log file format, CLI command syntax, etc. Clearly some of these things were hard to specify but we felt there was value in identifying them as interfaces so that a stake in the ground could be created as to their stability level.

Consider, things that appear obvious in one domain may not be obvious in another. For example, administrative CLI's in the middleware space tend to be unstable (i.e., change frequently) while CLI's in the OS space tend to be stable (i.e., change less frequently). This behavior appears to be driven both by market expectations and market maturity. In the application server space for instance the various vendors are still narrowing in on the right set of CLI commands for controlling application servers. In Solaris many of the administrative CLI commands go back to the early days of Unix, so changing them would potentially break a large number of shell scripts. This might not be obvious to people moving between the two domains so we capture the rationale as part of the decision.

Remember that all of this was put together in the late eighties and early nineties when the company was much more centralized in California and before the web. Although we do find the process still works irrespective of all the advances since then.

Similar to what you see in the rest of the world certain decisions become the bedrock of future decisions and are referenced constantly. For instance, we have a case opinion referred to as 1991/061 (the year 1991, the 61st case that year) that declares that an API can only be delivered into the system by a single package. From that case springs a set of constraints (and benefits) that are visible through the whole system. As you might expect most members of the ARC are familiar with this case's concepts.

In order to avoid a large startup surge we made the conscious decision not to review all of the legacy but instead concentrate on new projects and the delta's from old projects. Consequently, there may be large parts of the system which the ARC process may know little about but those areas have been stable for many years and we saw no benefit in spending the time to document that stability. As new ARC's were created they followed the same general philosophy: when someone planned to make a change, only review the area that was being changed. For practical reasons reviewing just the changes might require an understanding of the product as a whole. If that was required then that is what we did rather than forcing the reviews to be narrow and potentially missing an important interaction between system components.

One interesting decision we made was to review what we call projects as opposed to products. A product is composed of a collection of projects. By disconnecting the review process from the product creation process it allowed for flexibility in how products were assembled and re-assembled. For instance, we might review a system management project that might be delivered in several different products. By not tying the project to a product we only need to review it once before it is shipped the first time and not the subsequent times unless it changes. The major benefit of this approach is that things that do not change from release to release do not have to be reviewed every time. In fact the review process is designed to concentrate on the deltas between releases once a baseline has been set.

Definitions

Before any viable process can be proposed or evaluated, the critical terms need to be defined. The importance of establishing a "Lingua Franca" cannot be overstated. In the absence of clear definitions, discussions often degenerate in ways reminiscent to the Biblical rendition of the "Tower of Babel".

It is our experience that the core terms need to be defined with great attention to detail. Our internal definitions tend to read as though a squadron of attorneys had their way with them (although no actual legal review has occurred). If they aren't defined with such rigor, consumers tend to read their preconceptions and desires into the definitions. In the interest of simplicity and brevity, the following are only summaries of the relevant definitions we actually use.

Interface:
Any discernible behavior between two independent software components. Usually, this is limited to intended or documented behavior (possibly only internally documented).

Public Interface:
An interface intended for use by parties both external and internal to Sun Microsystems. Note that Sun has several sub-categories of Public Interfaces, each with differing Stability expectations.

Private Interface:
An interface whose supported use is limited to Sun Microsystems or a subset thereof (see Product/Consolidation/Project below). Note that we have numerous sub-categories of Private Interfaces, each with differing Stability expectations and usage domains.

Release Taxonomy:
A definition of release types (Major, Minor, Micro/Patch) and the level of compatibility with previous releases a customer can expect.

Interface Stability:
The level of compatibility with previous versions a consumer can expect from an interface and the release level where that expectation may be broken.

Stable:
An Interface Stability level assigned to Public Interfaces indicating a commitment that they will not change incompatibly (except in a Major release).

Unstable:
An Interface Stability level assigned to Public Interfaces indicating a commitment that they may change incompatibly in Major and Minor releases.

Volatile:
An Interface Stability level indicating that change may occur at a time.

Product:
A software bundle available to the consumer (it is on the price list or publicly available for free acquisition).

Consolidation:
A collection of software components built and tested as a whole and delivered to an integration organization in binary form.
Examples are the Solaris components SunOS, j2se (Java), CDE and Sun-Gnome.

Project:
An identifiable subset of a Consolidation typically associated with a feature or a specific change in behavior.

Note that many Sun products (notably Solaris) are never expected to produce a Major, incompatible release, so the Major number has been dropped from the product name. Solaris 10 is technically a Minor release.

Current state

The current architectural review process is defined as part of the larger Software Development Framework adopted by Sun in the early 90's. Most parts of the SDF describe activities that developers would expect follow, with perhaps an unexpected emphasis on producing prototype implementations as part of the design phase.

Under the SDF, review of architectural specifications is required. This review is accomplished by Architectural Review Committees,
(ARCs). Under the SDF, these reviews happen after a prototype was constructed but before the actual implementation begins.

There are multiple ARCs to accommodate the load and to match the expertise of the committee members to the projects under review. The domains of the various ARCs are delineated by horizontal lines drawn through the typical software stack. The domain is not determined by product definitions or boundaries. Drawing the lines horizontally facilitates the matching of members' expertise to the projects under review and the identification of similar issues across multiple products. Reviews are not broken up to fit the domains of the ARCs. When a project's domain spans that of multiple ARCs, the single ARC that is a best fit in expertise is used for the entire project.

The exact number of ARCs and their domains has changed over time from and initial five to a low of two to the current seven, although only four of the current ARCs have expansive domains.
Those four are:

- PSARC - Platform Software (SunOS, Networking, ...)
- LSARC - Layered Software (Development Tools, Desktops, ...)
- WSARC - Web Services

SHARC - Storage Systems

It is deemed very important that the committee is made up of actual contributors rather than professional reviewers. Actual contributors appreciate the tradeoffs that often must be made in product design, thus minimizing any tendencies toward pursuing perfection. Committee members also have "day jobs" with ARC participation expected to take 20-25% of the individual's time. It simply takes time to read and digest the project specification materials. The committee members are expected to provide a "systems view", rather than be specialists in any given specific project.

ARCs review projects, not products (although sometimes they are one and the same). A project is best thought of as a change or addition to an existing product. The reviewed content of Solaris is the aggregate result of the review of hundreds (perhaps thousands) of projects. A project may be as small as a simple bug fix, or as large as the addition of a major subsystem. Projects which don't alter interfaces visible outside of the project domain, such as most bug fixes, don't require ARC review.

Projects propose architectures. The ARC reviews those architectures. It is assumed that those associated with the project are the most capable of architecture and design with the committee members most capable of observing conflicts with other systems, best practices or internal standards. As one might expect from this decomposition of responsibility, ARC reviews strongly focus on the interfaces exported and imported by projects.

ARC reviews have a "shrink to fit" nature. Major or controversial projects may require a series of formal review sessions and the analysis of reams of documentation. Minor and obvious projects may be handled by a fast-track process, implemented entirely through e-mail, where approval is the assumed result after the passage of a week with no objections.

ARC approval is accompanied by a written opinion, which is reviewed by not only the ARC issuing the opinion, but all ARCs. Opinions serve a number of purposes. Some of the most significant are:

- Provide a clear map to what was approved
- Provide rationale for decisions, which over time becomes a valuable resource akin to "case precedence" in the legal system. Do you really want to repeat the same discussion multiple times?
- Provides a mechanism to express desires for future directions to both the project team and to management. Some of these items may be required while others may just be recommendations.

All case materials, opinions, meeting minutes and e-mail discussions are archived. This has proven invaluable.

A final observation is that less than 2% of the projects proposed are Denied. We use the term Denied not only because it is less pejorative that Rejected, but is intended to reflect that the proposal was not appropriate rather than fundamentally unsound.

What works

Sun's Architectural Review Process has been very successful in delivering the following:

- Interfaces and their associated stability are clearly defined. This is of significant value to not only the project team, but to all other project teams which may wish to consume those interfaces or supply similar interfaces. This enables the independent delivery of projects and products while minimizing the risk of breakage. If breakage occurs, it makes it clear where the responsibility lies and what needs to be fixed.
- Conceptual ownership of interfaces is clearly defined. Note that his is by project area rather that the names of individuals.
- Duplication of effort is minimized.
- Commonality in design and implementation is maximized.
- Repeating the same "mistake" is minimized.

What doesn't work (or work as well as we would like)

There are some expectations of the Architectural Review Process which have only been partially realized:

- As the process is passive, global system architecture is often not addressed. This is probably best addressed by another process. This is mostly a problem of expectations. The ARCs are not "the droid you are looking for". Often this function is performed by the ARC members, but not directly as part of the ARC process.

- Best practices are probably best created by the committee members as they have excellent visibility into the problems to be addressed. This does happen, but usually as a single committee member taking the lead and doing most of the work. This tends to only happen after enough divergence in design and implementation has happened to make the problem "hurt". It would be better to be proactive in this area.

- Failure of project teams to interact early in their development cycles often turns small problems into major issues. "ARC early, ARC often" has become an internal motto, but it is often unknown to first time participants in the process.

- The ARC process does not apply well to the review of hardware architecture.

- It would be too strong a statement to claim that the ARC process fails in the review of the absorption of community software into Sun's aggregate products such as Solaris and JDS, but missteps have been taken and the process needs to be enhanced or modified to facilitate such reviews.

- It has been difficult to quantify the contribution of the ARC process to software quality and product success. This is largely due to the "disasters avoided" nature of much of the contribution.

What is GNOME model?

With the assistance of Sun employees who are members of the GNOME community we have extracted the following key points from the GNOME process relative to understanding the architectural implications of changes to GNOME.

Some modules within GNOME have implicit stability guarantees (glib/gtk/pango), and the maintainers of those modules make efforts to ensure ABI compatibility. The GNOME community has automated tools to help ensure reasonable evolution of the API. Not all maintainers follow the same practices and other parts of GNOME are Unstable or, more properly, Undefined.

When an ABI is broken, and people complain, the GNOME community is good about fixing the problem and making a new release. In some situations, where only a handful of applications are affected by a change, the breakage was allowed to stay and the applications were asked to update their interfaces. This has been done in situations where the ABI breakage was related to a code fix that added significant value.

There is also the GEP process. It has been used in the past to help maintainers who have provocative ideas to work them out with the community to decide on the right approach. A description of the process can be found here:
http://developer.gnome.org/gep/gep-0.html

Our understanding is that currently the GNOME community is not using the GEP process although it does appear to have a number of good attributes relative to the Sun ARC process.

It is clear that there are a number of GNOME specifications available on various website. However there may not be central coordination point for pulling these together. So, our feeling is that there is a lot of good material out there available to members of the community who know where to look for it.

Proposal for GNOME

Gnome is important to Sun. Sun's current in-house desktop is CDE (Common Desktop Environment). It was developed in the early 90's, so it is getting a bit dated. Sun looks to Gnome to provide its desktop of the future (With the release of Solaris 10, the future may be today.)

Sun desires to hone its ability to nurture symbiotic and supportive relationships with freeware organizations in general. This is not only due to the increased importation of freeware components into Sun's software stacks, but also due to a desire to develop an active and constructive community around the recently released Solaris sources.

Added together, these two reasons make Gnome the ideal partner for Sun
to work with to develop development processes which interact well with each other. We are presenting our processes not with the idea that the Gnome community will adopt them, but only with the hope that the Gnome community will profit from our experiences and the result will be better architectural interactions between the two organizations.

First and foremost, we would like to work with Gnome to develop a common terminology, a Lingua Franca if you will, to characterize interface stability levels and usage domains. Sun's Interface Taxonomy is probably too complex for Gnome to adopt directly (many of its attributes are only relevant in a corporate entity), but individual taxonomies with a clear mapping from one to the other (and common terms when appropriate) would greatly enhance communication.

We would also like to encourage the Gnome community to develop a process which minimally owns the responsibility for deciding the association of stability levels with interfaces. The process would also ensure that the results got documented in a centralized location. For example, a web site where all results were announced and acted as the starting point for understanding the decisions that had been made across the community.

The CTO office is currently in the process of reevaluating its architectural review processes. There are several reasons for this, but the major impetus is to modify the process to better interact with freeware communities such as Gnome. As much as we hope the Gnome community can profit from Sun's experience in creating architectural review processes, we hope that Sun can learn from the collective experiences of the Gnome community in refining its own processes. Shared experience should result in processes which "Play Nice" with each other to the benefit of all.

To collect feedback on these ideas we've set up an alias at Sun called:
guadec-2005-feedback@sun.com where the authors can respond to feedback from the community.