Enabling technologies and constraints for software sharing in large astronomy projects

Gianluca Chiozzi*[a], Alan Bridger[b], Kim Gillies[c], Bret Goodrich[d], Jimmy Johnson[e], Kevin McCann[e], German Schumacher[f], Steve Wampler[d]
[a]European Southern Observatory, Garching bei Muenchen, Germany
[b]UK Astronomy Technology Centre, United Kingdom
[c]Space Telescope Science Institute
[d]National Solar Observatory
[e]W.M. Keck Observatory
[f]National Optical Astronomy Observatories, La Serena, Chile

ABSTRACT

The new observatories currently being built, upgraded or designed represent a big step up in terms of complexity (laser guide star, adaptive optics, 30/40m class telescopes) with respect to the previous generation of ground-based telescopes. Moreover, the high cost of observing time imposes challenging requirements on system reliability and observing efficiency as well as challenging constraints in implementing major upgrades to operational observatories. Many of the basic issues are common to most of the new projects, while each project also brings an additional set of very specific challenges, imposed by the unique characteristics and scientific objectives of each telescope. Finding ways to share the solution and the risk for these common problems would allow the teams in the different projects to concentrate more resources on the specific challenges, while at the same time realizing more reliable and cost efficient systems. In this paper we analyze the many dimensions that might be involved in sharing and re-using observatory software (e.g. components, design, infrastructure frameworks, applications, toolkits, etc.). We also examine observatory experiences and technology trends. This work is the continuation of an effort started in the middle of 2007 to analyze the trends in software for the control systems of large astronomy projects[1].

Keywords: observatory software, software sharing, technology trends, infrastructure frameworks, control system

1. INTRODUCTION

Software reuse has become a key factor in industry due to its potential benefits and savings. This paper explores what software reuse can mean for the astronomical community. To do so we define what we mean by software sharing, we explore the benefits and barriers to reuse and the technology enablers. We examine how reuse can be classified and provide real world examples by looking at the current state of affairs to determine how software reuse is practiced today across observatories. We explore how feasible reuse solutions might be at the various levels and analyze our domain to see where standards and specifications could be developed, common practices put in place, and where shared middleware and other assets could ultimately be deployed.

In "Measuring Software Reuse" [2] a typical application is decomposed and analyzed and the author shows that there is potential for 85% of a solution to be reused. This illustrates the great potential for software reuse assuming we can overcome the barriers. The intent of this paper is to foster software sharing across the observatories and hopefully to lay the ground work for broader scale software reuse in the future.

2. WHAT DO WE MEAN BY *SOFTWARE SHARING?*

Many papers on software sharing have been published. What most have in common is that software sharing refers to the utilization of *software assets* for implementing or updating systems where software assets are not just source code, but

* gchiozzi@eso.org; phone +49-89-32006543

can include all deliverables of a software development lifecycle: requirements, specifications and design, software components, documentation and test suites[3].

The concept of reuse is broad ranging from ad-hoc cut and paste reuse to formal reuse that requires significant planning and support. It is important to recognize and understand the different types of reuse as each have different *return on investment* (ROI), benefits and barriers in the short and long term. For the purposes of this paper we will categorize software sharing under the following: black box or white box reuse, opportunistic or planned reuse, horizontal or vertical sharing.

Sharing can occur at incremental levels of sophistication starting from opportunistic reuse to engineered system wide reuse. There are pros and cons to each approach. Typically the largest reuse benefits come from a systematic planned approach but we will need to see what makes the most sense for our community. We may find that a hybrid approach provides the best compromise between investment, initial flexibility and the ease of maintenance.

2.1 Black box or white box reuse

Black box reuse refers to reuse with no change. This is an ideal situation but often not practical because of limitations in the software asset or of a poor fit between the asset and the application. Typical examples of good black box reuse include math libraries, *slalib* (http://www.starlink.rl.ac.uk/star/docs/sun67.htx/sun67.html) and many other 3rd party libraries or components.

White box reuse refers to reuse with change. There is significant risk in this approach if changes are not incorporated back into the shared asset, since it becomes very difficult to follow up and profit from the improvements implemented in new versions of the original asset. Many open source projects have dealt with white box reuse by forking the original source code; the two versions may eventually be reunited with a larger feature set, or diverge due to different goals or expectations.

2.2 Opportunistic or planned reuse

Opportunistic reuse is also known as *ad-hoc reuse*. Almost all organizations and individuals practice ad-hoc reuse by salvaging old assets in some fashion. One of the more common forms of ad-hoc reuse is the cut-and-paste technique which is very flexible and inexpensive initially but extremely costly in later maintenance.

Planned reuse tends to be initially expensive and can be functionally restrictive but is inexpensive to maintain. Planned reuse can be further sub divided into evolutionary and systematic. Taking an evolutionary approach, users are encouraged to identify all potential reusable items and to make them available to others in the organization without taking up front an explicit effort to make them reusable. Refactoring is encouraged at each reuse. This approach introduces minimal overhead costs on the side that makes available assets for reuse, since each asset is only extended when there is a real need. This in particular avoids high costs in generalizing items that are supposed to be reusable but that in practice will never be. Traditional systematic software reuse is a highly structured approach with designers and developers identifying potentially reusable components upfront and developing them in a generalized way, ready for reuse[4]. This approach requires significant planning, built in abstraction and other hooks. It is striving for black box reuse but there is a cost associated with this.

2.3 Horizontal or Vertical

Horizontal reuse is the use of generic components or libraries across a wide variety of application and domain areas. Typical examples might be regular expression or mathematical libraries. Usually such libraries are not specific to astronomy and we all make extensive use of such software. If we consider the domain of applications related to astronomy, a generic positional astronomy library such as *slalib* should also be considered as a very good example of horizontal reuse: it addresses a very general problem common to most astronomical applications and it is easily reusable because self-contained and independent from any external constraint.

Vertical reuse is intended here as the use of software within a similar functional area, typically involving fairly large, or complex components that provide a complete solution to a domain-specific problem. For example, the SkyCat and JSky tools (http://archive.eso.org/skycat/) provide an off-the-shelf solution to the problem of combining visualization of images and access to catalogs and archive data for astronomy. Another excellent example of vertical reuse is the Tpoint package (http://www.tpsoft.demon.co.uk/), used on most telescopes worldwide to evaluate pointing models both as a stand-alone application and integrated in the control systems. A generic Telescope Control System or a generic

Observation Preparation Tool usable by different observatories would represent a valuable example of vertical reuse. But it is much harder to achieve vertical than horizontal reuse because the integration of such a complete solution requires accepting on one side the requirements and the conceptual assumptions that have driven its design and on the other side the technological choices that have been done in its implementation.

3. BENEFITS OF SOFTWARE SHARING

There is strong consensus among software engineers that software sharing can bring major benefits to the projects. In practice though, such benefits are not always easy to achieve or to measure. This often makes it difficult to convince project stakeholders to invest in software sharing. The benefits can include:

- Improved productivity and quality while decreasing the cost of development.
- Saving time and money while achieving greater reliability
- More effective use of domain experts
- Building a knowledge base that improves after every reuse.
- As new systems are produced, improvements in the base architecture and common core are propagated throughout all systems
- Standards can be developed and deployed including interface descriptions, programming languages, operating systems, networking etc.
- Reduced maintenance costs, since maintenance costs for common parts are shared and less code produced also means less code to maintain.

A number of papers and articles have focused at different levels on a formal measurement of the benefits of software reuse. An article published on DDJ in 2005 [4] shows that systematic software reuse has the potential to be highly negative if the assets that are built are not quickly reused on multiple projects. Systematic reuse does, however, have the highest slope in the early days, meaning that it can provide a very quick return on investment if properly implemented.

Evolutionary reuse starts off with low incremental benefits to the organization but quickly begins to generate increasing value as content is re-factored and made available to end users. It provides a nice compromise for companies and organizations looking to enhance the benefits they are currently getting from their ad hoc reuse efforts but who are unwilling or unable to invest the time required to set up and manage a structured systematic reuse effort.

Finally, ad hoc reuse currently generates modest benefits to an organization and it will continue to do so, although these benefits grow slowly and are far from being optimized.

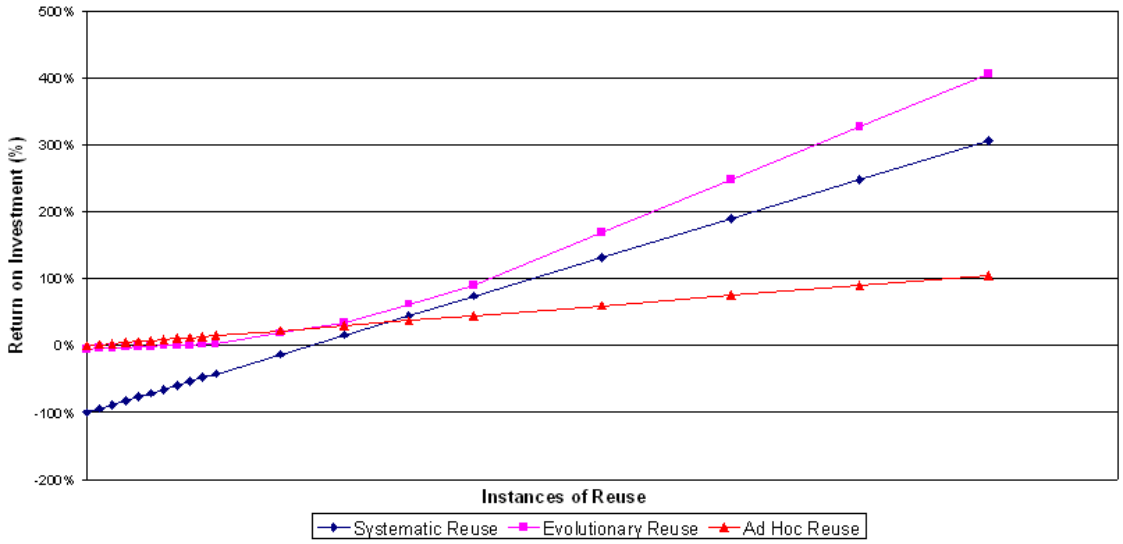

Fig. 1. Costs and return on investment of code reuse (reproduced with permission of CMP Media LLC)[4]

We can see a number of good examples within our organizations of internal systemic reuse. Two such examples are the Keck twins and the Telescope Control Software (TCS) developed for the VLT Unit Telescopes (UTs). Keck is an

interesting case study. Although the two telescopes were supposed to be virtually identical, the actual initial implementations contained many important differences principally because of the 5 years that separate the two systems. Initially Keck I was VAX based with a large collection of proprietary solutions. By the time Keck II came around, five years later, technology had significantly changed as well as the observatory's realization for the need for better system engineering. Keck II, which was based on EPICS (http://csg.lbl.gov/EPICS.html), was designed such that it could be easily retrofitted to Keck I utilizing key elements such as the KTL that already provided a good abstraction from the underlying implementation to allow reuse of Keck I UIs and other infrastructure. Using a framework such as EPICS, designing for reuse and having appropriate levels of abstraction meant that the final solution for Keck II was easily retrofitted to Keck I to the extent that both telescopes share well over 90% of the same code for the telescope control system.

On an even larger scale the VLT TCS software from the beginning was conceived to be reusable on different telescopes. All Paranal Telescopes: Unit Telescopes (UTs), Auxiliary Telescopes (ATs), VISTA and VST and several other ESO telescopes are reusing the basic architecture, design and code of the VLT UTs. The following figure shows how, using the UT TCS base, the number of lines of coded needed to implement each telescope is about 1/3 of the UT reference case. It should be noted that from a TCS perspective none of the developed telescopes is significantly less complex than the reference implementation and the hardware across these telescopes is often substantially different. This analysis shows considerable savings in the effort needed to implement a new TCS when VLTSW and the VLT TCS were used as starting points.

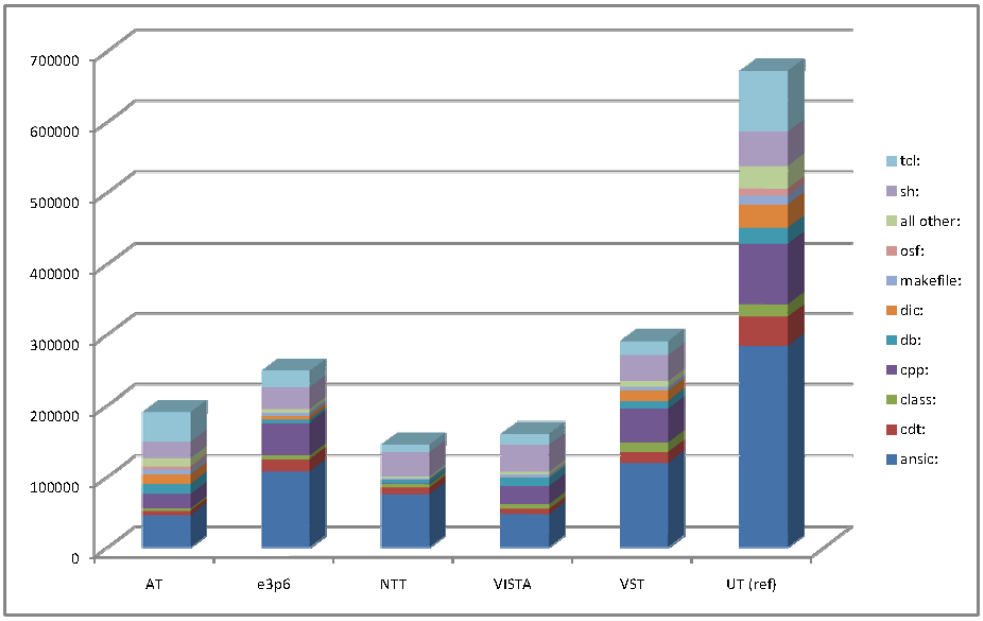

Fig. 2. Lines of code for the TCS of ESO telescopes

In significant systematic reuse cases such as these, there are other substantial savings beyond reduction in code and effort. Analysis and design is simplified and improved whenever systems are sharing the same architecture and high level design, as well as the same standards and framework. The project teams were free to concentrate on the peculiarities of each telescope and less time had to be spent on the general and common aspects. There is also an implicit confidence factor derived from the fact that the core software has already been running for a number of years on the already operating telescopes. The adoption of a common architecture, design and infrastructure also reduces the maintenance and operation costs, because the same team can take care of each telescope while requiring the knowledge of only one high level system.

The laser traffic control system on Mauna Kea is an interesting example of a cross-organization software sharing at the application level. The Laser Traffic Control System (LTCS) [20] is a component of the Keck Observatory laser safety system dedicated to the problem of laser beam avoidance, using both static and dynamic geometry to predict "collision" events between one or more lasers and one or more telescopes. It is a joint development project of Keck, Gemini,

Subaru, and the William Herschel Telescope, with support from participating observatories including Canada France Hawaii Telescope (CFHT), NASA's Infra-Red Telescope (IRTF), and the University of Hawaii. Several other observatories in Chile and Hawaii are currently investigating reuse potential as well. Mauna Kea is in the process of an operations transition from a "lasers yield" to a "first-on-target" LTCS model, which has required some collaborative LTCS development in the last year.

A key challenge will be how to further advance and extend similar reuse between different organizations and observatories. What might the specific benefits be to the whole astronomy community? The astronomy community is small, distributed and very specialized and a coordinated effort in software sharing might bring specific advantages, even though some of the development benefits described above might be more difficult to achieve in a cross-organization environment. Taking a broader view on the benefits of software sharing across observatories we can imagine the following:

First the end users, i.e. the astronomers, may begin to see observing tools looking more "alike" and see increased interoperability between tools. This will save them time and effort in understanding the particular "quirks" of any given observatory. Observatory staff and observatory operations might also see benefits. Standardization on data collection, reporting and analysis tools might make it possible to compare the performance of different facilities and identify the areas for improvements in a meaningful way. Personnel could easily move from one facility to another and be quickly highly productive, facilitating exchange programs and collaborations between the technical teams of different organizations. The few companies providing consultancy and outsourcing for our domain could concentrate on a much smaller set of technologies and be able to provide their services to a wider set of observatories, with a shorter startup time and, most probably, at a lower cost for the projects. This has been demonstrated in the High Energy Physics community, with the wide adoption of EPICS

4. BARRIERS TO SOFTWARE SHARING

There are many barriers to software sharing. This is particularly true for cross-observatory sharing, where different sites will have different technology solutions. Common barriers include differences in functionality, programming languages, target and operating systems, standards, and data format. Other barriers are often institutional, such as timing between projects, the ‘Not Invented Here’ syndrome, and the cost and politics of sharing development with other sites. The latter is often seen as a major barrier: reuse is not free. It has been shown that reusable components cost about 50% more to develop than one-off components [2].

Few of these barriers are purely technical, like programming languages or target OS differences. The most important ones are actually more sociological or related to project management processes and practices. Perhaps the largest obstacle to software sharing across astronomy projects is with the astronomical software community itself. It is a fact that, with a few exceptions, there is no significant track record of reuse and currently no significant mechanisms to promote or support it. A key barrier to overcome will be to actually build this software community. Right now any reuse elements we choose will be on the order of design philosophies and methodologies. Other reuse elements would be chosen by each project based upon some serendipitous alignment of design or implementation strategies.

In a small community like ours, timing between projects is probably the highest barrier to sharing initiatives. In order to start a successful sharing activity, there must be a critical mass of potential users with very similar needs. This is apparent, for example, in the Linux development community, where the very high number of users makes it easy to form groups looking for the same features. In the telescope control community there are very few projects at any one time in the same development stage (in particular in the initial design phase) and therefore it is very difficult to create a community seeing an immediate advantage in sharing software. One can imagine that existing facilities would be hesitant to realign their efforts while new facilities would look for advanced systems they could leverage. The timing issue often shows up even within the same organization where different internal projects do not manage to share software because they start at different times. A typical reason to adopt a different solution is the diffusion of newer technologies that promise to offer much better results than the ones adopted by the previous project.

But if our community grows such that more systematic reuse can occur, then cost may become a significant factor as individuals will need to provide more time helping to define standards and specifications, provide reference implementations and provide other support functions. At that point, political support from project management will become the most important obstacle to overcome. It will be a task for the whole community to demonstrate that the additional upfront costs will be paid back in the following years.

5. FOSTERING REUSE

It is hoped that this paper and associated round table discussion will provide a foundation that will assist in setting standards for the common areas of astronomical software control. By building this foundation we can effectively be pooling our current expertise and future development costs. The High Energy Physics community has shown that large scale software sharing is not an impossible task. Their community, while larger than ours yet still small in relation with many huge open-source communities, has been able to put in place major software sharing enterprises like EPICS. The timing seems right for the astronomical control community, since there are a number of major projects starting or in their initial design phases.

There are multiple facets to fostering reuse and software sharing. These consist of some well established patterns and best practices adopted by most open source initiatives, interest in the community and management sponsorship to promote and maintain reuse (depending on the availability of sufficient funding) and possibly working groups to provide leadership on defining domain specific standards, specifications, reference implementations and middleware homogenization.

5.1 Open source practices

Rather than detail the individual practices, we only introduce some of them here with a view to returning to them when we reach a more mature stage in the exploration process.

Charles Leadbeater has identified five basic principles that successful community projects share [5]. These principles are based upon a solid *core*, a clear understanding of the process of *contributing*, a means of effectively *connecting* the community, an understanding of *collaboration*, and a recognition of the *creative* nature of the process.

These principles, and other similar ones, all provide insight as to what makes some open source projects successful while others fail or under perform. Any number of projects can be examined with respect to these principles. We would do well to understand the implications of adopting these principles and how best to follow them when fostering software reuse and sharing at observatories.

5.2 Community interest and management sponsorship

The International Virtual Observatory Alliance (IVOA, http://www.ivoa.net/), the National Virtual Observatory (NVO, http://www.us-vo.org/), the Virtual Solar Observatory (VSO, http://vso.nso.edu/), and other Virtual Observatories (VO) are good examples of astronomical collaborative software projects, with many diverse groups and people involved in developing standards and software in each. However, these VO communities have a fundamental advantage in that they each provide a product that is *directly* visible to the 'customers' (scientists). So there has always been a strong interest and support for VOs from outside the software development community. In particular, though the various national VO projects are funded independently, most were funded on the assumption that they would collaborate, with the associated travel and effort costs involved.

Observatory software, on the other hand, is much less visible to the scientists. So, while many developers can recognize and understand the potential benefits of software reuse, there needs to be an effort to engage the 'customer' base of observatory software. An important component of this base is *observatory management* - many of the benefits relate directly to the development and operational aspects of modern observatories.

Getting management "buy-in" goes some way to addressing the "cost"/"politics" barriers. This is a hard problem, as the development management team may differ from the ultimate operations management team. This problem is particularly acute with the planned systematic approach to software sharing. It is difficult to engage the interest of the first few development management teams when there is an actual negative impact - despite the long-term payoffs that can be realized during the operation/maintenance/upgrade cycle that always follows. It takes some collective foresight on the part of management to fully appreciate these long term gains. Nevertheless, it is possible.

Because few individual observatories have the software personnel resources to dedicate significant resources to developing reusable software, some significant collaboration is required. In addition, such a collaboration would produce reusable software that is shareable across a wider range of projects as well as more readily take advantage of the diverse knowledge and skills available in the observatory software community at large.

A possible approach may be found in funded international collaborations, where it is already accepted that the collaboration aspect involves initial extra cost, but is the trade off for the project happening at all. In the case of a software collaboration, the cost savings on future projects and on future upgrades may be considerable.

An important step in this direction has been taken by the ALMA collaboration. Leveraging on the fact that the deliverables of the project had to be shared among all partners, it has been decided with the support of management to make available all software under an open source license. A range of projects of medium size have profited from this opportunity and have adopted the ALMA Common Software (ACS) infrastructure [12]. The ALMA project provides support through community meetings and a mailing list and receives in return valuable feedback and bug reports. ALMA management also views this model favorably because the impact on cost is almost negligible. Still, this collaboration is too "asymmetric" because the external projects, being much smaller than ALMA, do not have the resources to provide a major active contribution to the development of ACS. To make this kind of collaboration really flourish it would be probably necessary to have at least two "equal partners" as it is now happening in the synchrotron accelerators community with the TANGO project (http://www.tango-controls.org/).

5.3 Working groups

One successful approach to fostering reuse is to organize a group with the task of creating a project that meets the needs of the community and a plan for implementing the project. Initially this group of interested participants would meet to define the project, possibly in terms of a system architecture that identifies areas where sharing can occur.

A project of this size takes the work of many people. Typically these projects are setup as working groups operating within a well-structured organization under an agreed upon process. The output of these groups steers the strategic and technical aspects of the project. Deliverables include the definition of the project itself, standards, and possibly the components of the architecture. The IVOA process for the creation of standards, involving working groups at multiple levels, is shown in Fig. 3

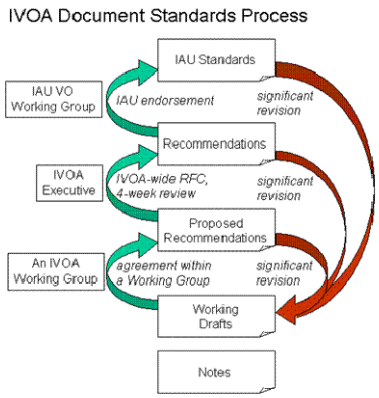

Fig. 3: IVOA standards creation process includes steps for community review and revision

The creation of a project of this scale focused on sharing will present many challenges and a tremendous amount of time and effort from the working groups to explore the various options and help define solutions. But providing an opportunity like this may be exactly what is needed to overcome the barriers to reuse. A well-defined, exciting project has the potential of engaging not only the astronomy development community at the large observatories and projects, but also the larger community of skilled programmers who would be drawn to a project with a potentially profound impact on man's knowledge of the universe.

5.4 Getting people involved

No collaborative software process can survive without the involvement of interested supporters. Management, staff, and "customers" need to be engaged and actively involved in the process. While a few key people are essential to any project, collaborative projects must reach out to and profit from the creative inputs from others outside the core group.

Connecting interested parties at conferences and collaboration on papers have been shown to act as a convenient and powerful catalyst in promoting software sharing and re-use. The early collaboration between the Mauna Kea

Observatories and WHT on development of LTCS resulted from a chance encounter at the SPIE 2002 Astronomical conference [20]. This led to collaboration on evolutionary changes and a follow up paper [21] which in turn catalyzed wider sharing. This current paper grew from a collaborative paper analyzing software trends for ICALEPCS 2007[1].

Distributed projects must also work harder to maintain the interest, involvement, and interaction among people working in diverse locations, on diverse projects, with diverse goals. It is too easy for isolation factors to blur goals and impose barriers to understanding the contributions from others. Frequent meetings among small working groups are essential and the project would benefit from such novel actions as staff exchanges, sabbatical opportunities for software developers to join groups to make more significant contributions to the software sharing collaboration, and other methods of fostering communication and encouraging creativity.

An intriguing possibility is to look at federating with existing, related collaborations. There are certainly projects in related fields such as robotic telescope control and advanced amateur astronomy that may well be interested in software unification across the astronomical software spectrum. Such a federation would take some care and effort to set up correctly, as the various collaborative groups would need to be of similar size and scope to prevent the smaller groups from being dominated.

6. OPPORTUNITIES FOR SHARING

6.1 Software Engineering Standards

An obvious and straightforward possibility for sharing is in the area of software engineering standards. These set of standards range from coding and design to development and process. When projects develop these standards, they often duplicate other existing projects, whether by informal 'borrowing' or similar circumstances. By adopting a set of standards, taken from appropriate current projects, this sharing could be formalized and would result in long-term savings. A more active approach would be branding the adopted standards as an output of the astronomical software community, and issuing them explicitly under a Creative Commons license, available for reuse, refactoring, and extension.

Coding and development standards range from preferred languages to coding conventions to development infrastructure and to distribution. Although it would be impossible and foolhardy to confer approval upon only a small subset of programming languages, down-selecting for those languages where significant support exists would increase the limited available resources within the community. Similarly, the selection of specific code typographic standards, although reasonable within a project, may only need to be rigorously defined if software sharing extends into white box reuse. However, having preferred sets of coding standards, for instance published C/C++ standards [14], would certainly facilitate the eventual migration toward software reuse between projects.

In the same sense, sharing development and distribution environments encourages closer cooperation between projects. Development environments can vary from simple *make* or *ant* source builds up to sophisticated environments such as Eclipse. These environments not only support common code development practices, but also may help define build, test, and release practices for project sharing. Likewise project distribution practices help disseminate software more readily. Projects that share both build/release philosophies and release export mechanisms easily can share parts of projects, or subcomponents.

At a higher level, projects can share software engineering processes. Most astronomical projects are constrained by funding agencies and common practice to the usage of a classical 'Waterfall' design methodology [15]. As a consequence, software projects must follow a design path predicated upon conceptual, preliminary, and critical design stages, each culminating in a design review check point. Software projects often share resources by the exchange of reviewers in the check point review process, allowing software groups from one project to observe, participate, and comment upon work done by other groups. Since this process is beneficial to both groups, it would be useful to codify the work products and deliverables for these reviews. Some issues that may be resolved are the level of effort required for each of the conceptual, preliminary, and critical designs, the associated design details (e.g., UML use cases, sequence diagram, class and deployment diagrams), and the types of documentation necessary. By agreeing upon the engineering process standards, a community can not only better support its members during the review process, but can take advantage of similar work to jump-start their own design and development efforts.

6.2 Functional Interface Standards

Very many of the modern observatories' software architectures are based on a "principal systems" model where the control of the major observatory systems (telescope, instrument, data handling, etc.) are delegated to a major software component [6],[7]. This is such a common model that it looks possible to standardize the definition of the functional interfaces of at least some of these systems. For instance, although instruments differ significantly in capabilities, their software interface is often reduced to commands to initialize, load and/or set configurations, and take data.

For telescope control, the Portable Telescope Control System (PTCS) proposed back in 1997 [8] has been adopted by a number of telescopes, and could form the basis of a standard interface.

Discussing this topic we all agreed that the availability and wide adoption of a generic and portable TCS would be extremely valuable. What surprised us was finding that in practice there is no established generic TCS used widely across observatories, at least as far as we know. The PTCS project mentioned above dates back to 1997 and we could not find information about recent development. Other systems like the Talon project (http://sourceforge.net/projects/observatory) seem to have had an initial active phase and languished after that. A recent project started at the Universidad Technica Federico Santa Maria in Chile and presented at this conference [13] is still too preliminary to say how it will evolve.

We are therefore asking ourselves what are the reasons for this situation. Could it be that this objective is too ambitious and therefore attempts to it are doomed to failure? Before starting new sharing initiatives, it is necessary to perform an unbiased analysis of the real interest and of the real chances of success, to avoid dispersing resources in projects that will most likely not produce useful results for our community.

6.3 Data Interface Standards

In Observatory Software the major subsystems in particular often exchange or share information via data structures, frequently "hidden" behind the functional interfaces. For instance, an instrument may be given a "configuration", or read one from a database; in both cases the configuration has been created by a different component. Increasingly these data interfaces are described using a standard language, such as XML, and may be one of many "data models" that are defined by the observatory.

Although instruments are sufficiently different across observatories that sharing a standard definition of an "instrument configuration" is ambitious, the same is not true of many other systems, and it may be possible to agree on standards in other areas. Many modern telescopes probably have telescope configuration interfaces defined in some way. Given the publication of some of these, a way forward to combine them into a baseline standard might be possible. Some work in this area has already been done with the development of the Remote Telescope Markup Language RTML (http://monet.uni-goettingen.de/twiki/bin/view/RTML) developed for communication between (mostly) robotic telescopes and agents. Though its scope is somewhat different from that of more "conventional" ground based observatories, it could an interesting potential starting point. It does re-use virtual observatory standards, for instance for (space-time) coordinate expression.

Another potential area is the "Project Data Model" (explored in more detail in [9]). The PDM describes the observing that an investigator wishes to carry out. For instance much of the information input to a proposal to use an observatory is common and may be susceptible to a standard description. And surely such items as a user database can be standardized.

In this area the data handling community is well ahead of the control community. A common interface in astronomical data analysis - FITS - has existed for decades [11] and though the original FITS is perhaps more correctly described as an interchange format, the more recent, astronomy specific, formulations (http://fits.gsfc.nasa.gov/iaufwg/) can be viewed as data models. FITS has been a tremendous success, allowing a number of data reduction and analysis systems to be shared and, importantly, intermixed - enabling significant software reuse. It is simply accepted that any replacement for FITS has to be an agreed standard. "Rolling your own" simply will not work.

There is a barrier to agreeing these standards: with a model such as FITS the vertical reuse benefits are obvious and many. Similarly the VOTable standard (http://www.ivoa.net/Documents/latest/VOT.html) allows the sharing of astronomical tabular information across many applications. But is there such a big driver to adopt common observatory data models? Is there an obvious need to share an instance document describing where to point your telescope?

6.4 Middleware and infrastructure

We can consider middleware to be the plumbing between the OS and our applications. A software infrastructure provides common services that all applications need such as alarming, auditing, history etc. Middleware excels at hiding distribution, hiding heterogeneity, providing consistent and often (open) standard, high-level interfaces and common services. It should be noted that a common architecture typically implies standard data formats also and not just interfaces. Perhaps the greatest benefit middleware and infrastructure brings is that it allows us to focus less on the system internals and more on the applications. Writing middleware is difficult, time-consuming, and requires considerable expertise as well as resources for development and testing. We are all using middleware, sometimes rolling our own solutions, and all require a software infrastructure but we are not yet trying to standardize on a solution or provide standardized abstractions over our systems such that they can be easily reused or interoperated with. An initial effort aimed at providing a middleware-neutral infrastructure has been started by the ATST project but this work is still experimental [16]. Middleware and a software infrastructure make it easier to develop and evolve systems using reusable assets. CORBA continues to be one of the most mature middleware sectors with many reliable open source implementations.

A key question is: is it feasible for us to consider a common middleware or software infrastructure implementation? The large scale benefits might seem fairly obvious, but it is unclear if this would be practical or what the costs/effort involved would be. One solution might be to put together a group that looks at the existing solutions, both commercial and non-commercial, and explores future growth in this area. There are a number of middleware solutions in our space, including ACE/TAO, ACS, DDS, EPICS, ICE, RTC and TANGO. Four of these are CORBA based.

Infrastructure is broad and can include: User Interface Management, Data Access, Data Management and Publication, Data Modeling, Calculations, Workflow Orchestration, Security, Alarm and Event Generation, Application Settings, Configuration, Diagnostics History and Auditing and more. Each of these areas represents opportunities for sharing. One can imagine that there might be a set of core services and a range of frameworks, each covering a subset of the above. This has the advantage that we can tackle opportunities for reuse in a piecewise fashion treating the end solution as a broad collection of services and associated frameworks brought together in a way that allows application developers to exploit them in order to build applications that are well integrated with one another.

Given the right approach and resources, real progress is possible in many of these areas: in the process control world, OPC has been very successful at addressing many of these aspects incrementally adding Data Access, Alarm and Events, History and now an OPC unified architecture (UA) which are implemented on a services-based architecture, leveraging existing standards such as XML, SOAP and the WS initiatives. With it came a slew of additional packages from group members and third party suppliers that handle UI, gateway connections, history applications and more in a standardized uniform way. This is also a good example of how existing systems can be augmented, rather than reworked, to include new technologies allowing them to interoperate in a standardized way without having to rework the existing system.

Although middleware and infrastructure is a large area, what should be clear is that we shouldn't see this as an all or nothing solution. With the right level of abstraction, foresight and use of standards there is no reason why existing subsystems cannot be augmented and new (sub-)systems designed to leverage common services, and yet continuing to use information centric or service centric solutions that best suit their needs.

6.5 Software packages

There is a number of existing software packages in use. A few of them, like *slalib*, *cfitsio*, and *jsky*, are mentioned in this paper. It would be to the advantage of the software community to actively support and encourage these packages. There are software packages in use at the various observatories that may be recommended or promoted by the group to fill a necessary software niche. For example, the AAO DRAMA (http://www.aao.gov.au/drama) package has been used at several observatories with great success. LabVIEW user groups are active in the astronomy community, developing and sharing clever VI's and discussion of useful tools [http://www.physics.unc.edu/~cecil/science/remoteOps.pdf].

A good solution would be to create an Internet site where it is possible to list useful software packages by category, post reviews and rank the packages or request help in finding "the right package for the problem". Many user communities on the web have similar systems in place. The problem of such a system specialized for the astronomical software community would be to reach the critical mass to make it sufficiently active to be useful. A possibility would be to join a similar community, assuming that a number of packages might be useful not only for astronomy but also for other domains.

A very interesting initiative to look at is the *NASA Earth Science Data Systems Software Reuse Working Group* (http://www.esdswg.org/softwarereuse). Their community is from many aspects very similar to our own and their working group is discussing the same problems we are trying to deal with in this paper. There might be therefore good opportunities for collaboration, in particular for what concerns methodologies and tools to be used in the sharing efforts.

6.6 Virtual Observatory

The IVOA [19] is an example of a working, software sharing collaboration that has produced a number of standards, data models, protocols, and applications for science data sharing. Examples include the VOTable data access standard for tabular and catalog data, technology infrastructure standards like VOSpace and Service Registry, as well as data models like the Space Time Coordinate metadata standard. In addition tools produced by the various VO projects are rapidly adopted across the IVOA community. The IVOA provides a prototype for software sharing the observatory control community could follow.

Adoption, reuse, and integration of the products developed by IVOA offer a number of interesting opportunities for the control system community. Though it may be argued that these are targeted at solving a different problem, there is much in common: for instance, distributed information and services and security. The merger of VO and real observatories is already happening with projects like LSST and eStar. The publication of transient events as VOEvents and software agents such as the ones following up target of opportunity observing [17] will be an important facet of observing with these systems.

Finally, we can grow the VO infrastructure and connect virtual and "real" observatories. One clear opportunity here would seem to be the concept of a "phase 0" for observing — essentially the step before observatory proposal submission that involves identifying the need for (real) observing and then locating a service (facility) that can carry out that observation [18].

6.7 Designs, knowledge and experience

As noted previously in the context of designing software for the upcoming Extremely Large Telescopes [6], the top level (architectural) designs of many observatories are very similar. We know that the "Principal System" design used in Gemini (OCS/TCS/ICS) is a common approach (though under different names and with variations. With the extensive experience we now have of building and maintaining systems based around this architecture, and with the software budgets for ELTs likely to be constrained, is it not an appropriate time to formalize these as the standard approach to astronomical observatory control software?

It is also very likely that many low level, quite detailed, designs are going to be very similar. Because this is not seen as ground-breaking work, the details of the issues involved often languish unpublished. How do we share these in a practical way? Is there any scope for a library of domain patterns that could become a knowledge base for astronomical control software engineers?

Such a resource might also become a focal point for even experienced engineers openly discussing new (but possibly common) problems and their solutions. Such collaboration is surely a pre-requisite for being able to share software at a deeper level.

In the web 2.0 age it surely just requires the willing from the community. There is already much happening in terms of utilizing modern communication tools (wikis and social networking) in other areas of the astronomy community, within projects, in more specialized areas (for instance robotic telescopes, [10]), or among informal groupings. Can we find a good model to build on?

7. CONCLUSION

We all acknowledge that we do not want to develop yet another infrastructure and that we want to concentrate on applications development, where our domain specialization is unique; yet we continue to invest in divergent infrastructure despite having many areas of commonality across projects.

In this paper we have examined what we mean by software sharing, the benefits and barriers and looked at sharing within one organization, sharing when the solution is using the same platform & framework and sharing across a heterogeneous system. From this analysis, we hope to share lessons learned and identify areas where greater cooperation across current operational observatories, as well as facilities under development, would be beneficial. The move towards

open source paradigms and large international collaborations to share cost has made such collaboration more practical and affordable. In particular, we hope to be able to share technical architectural elements and infrastructure components more frequently, so that each observatory or project can focus on specific, science-driven needs.

Opportunities for sharing exist across a broad spectrum of areas. On an extremely ambitious scale we could achieve broad scale reuse across all these areas providing a solution that would allow us to take components that are built independently by different organizations at different times and assemble them to create a complete system.

The question is: how do we get there?

8. ACKNOWLEDGMENTS

The authors would like to thank the many colleagues in the astronomical observatory community who have given so freely of their ideas and time as we have developed this paper. We want to thank in particular Dave Silva and Anders Wallander for their participation and essential contribution to the discussions in our working group in the last year.

9. BIBLIOGRAPHY

[1] G.Chiozzi, K.Gillies, B.Goodrich, S.Wampler, J.Johnson, K.McCann, G.Schumacher, D.Silva - "Trends in software for large astronomy projects", 11th ICALEPCS Int. Conf. on Accelerator & Large Experimental Physics Control Systems, Knoxville, 2007

[2] J.S.Poulin - *Measuring Software Reuse: Principles, Practices, and Economic Models*, Addison Wesley, 1996

[3] Department of Defense. Software Reuse Executive Primer, Falls Church, VA, April, 1996.

[4] L.Amar, J.Coffey - "Measuring the benefits of software reuse", Dr. Dobb's Journal, Jun01, 2005

[5] C.Leadbeater, *We-think: The Power of Mass Creativity*, Profile Books Ltd, 2008

[6] K.Gillies, J.Dunn, D.Silva, "Defining Common software for the Thirty Meter Telescope", SPIE Proceedings 6274, 2006.

[7] S.Wampler, "The Software Design of the Gemini 8m Telescopes", SPIE Proceedings 2871, 1997.

[8] J.A.Bailey, R.M.Prestage, "Portable telescope control system project", SPIE Proceedings 3112, 1997.

[9] A.Bridger, B.Butler, "The ALMA/EVLA project data model: steps toward a common project description for astronomy", this conference.

[10] The Heterogeneous Telescope Networks Consortium, http://www.telescope-networks.org/

[11] D.C.Wells, E.W.Greisen, and R.H.Harten, ""FITS: A Flexible Image Transport System," Astron. Astrophys. Suppl.", 44, 363–370. 1981

[12] G.Chiozzi et al., "Application development using the ALMA Common Software", SPIE Proceedings 6264, 2006.

[13] R.Tobar, H.Von Brand, M.Araya, J.Lopez, "An Amateur Telescope Control System towards a Generic Telescope Control Model", These proceedings.

[14] H. Sutter, A. Alexandrescu, *C++ Coding Standards*, Addison-Wesley, 2004.

[15] W. W. Royce, "Managing the Development of Large Software Systems," *Proceedings of IEEE WESCON* 26 (August): 1-9, 1970.

[16] S. Wampler, "A middleware-neutral common services software infrastructure", 10th ICALEPCS Int. Conf. on Accelerator & Large Experimental Physics Control Systems, Geneva, 2005

[17] A.Allan, I. Steele, R.White and F.Hesseman, "Autonomous Observing: The Astronomer's Last Stand", Proceedings of ADASS 2007, in press

[18] K.Gillies and S.Walker, "Design for a phase 0 network", SPIE Proceedings 4844, 2002.

[19] The International Virtual Observatory Alliance, http://www.ivoa.net/

[20] D. M. Summers, et. al, "Implementation of a Laser Traffic Control System supporting Laser Guide Star Adaptive Optics on Mauna Kea", *SPIE Proc.* 4839-57, 2002

[21] D. M. Summers, et. al, "Second-generation laser traffic control: algorithm changes supporting Mauna Kea, La Palma and future multi-telescope/laser sites", *SPIE Proc.* 6272-143, 2006