

ODI Software Plan

For

Tier 0 and Tier 1 Software

23 June 2008

1. Introduction

This is the Software Plan for ODI Tier 0/1 software. ODI's Tier 0/1 Software will provide the capabilities required to make the ODI camera useful to the scientific community. These capabilities include collecting, processing, and storing image data from the 64 OTAs on the ODI Focal Plane. This plan describes the ODI Tier 0/1 Software and how it will be implemented. Members of the software team will use this plan as a guide to their activities and the ODI Project Manager will use it to monitor progress of the project. This plan provides a brief overview of the ODI Software Plan in Section 1. Section 2 describes the software organization and Section 3 describes how the project will be managed. Section 4 provides a description of the ODI Software Functional Requirements and a brief description of the Service Oriented Architecture chosen to implement the functional software requirements. Section 5 describes the technical tools and software development processes that will be used to implement the software. Section 6 describes the software milestones and provides staffing, budget and schedule information.

In addition Appendix A, "ODI Software Requirements, Deployment and Services" provides: 1) more detail information about how the software functional requirements are mapped onto the "ODI Services" that will be used to implement the required functionality, 2) figures that show where the "ODI Services" reside in the hardware architecture, and 3) a more detailed description of the "ODI Services".

1.1 Project Overview

The Tier 0/1 software provides the necessary capabilities to 1) prepare observation scripts, 2) perform observations, 3) make quality assessments with a "quick look" image display, 4) store the raw image data, 5) perform Tier 0/1 data reduction, and 6) store the Tier 0/1 reduced data in the NOAO archive. A detailed list of requirements for ODI software is located in Paragraph 4.

The project includes the analysis of the software requirements, development of hardware/software architecture, design of the software, implementation of the applications, integration with other system components, and testing and verification of the integrated ODI system.

The hardware necessary to host these software operations will be designed, procured, and installed as part of the ODI Tier 0/1 software project.

The objective of the Tier 0/1 Software Project is to provide astronomers with the ability to collect, store, and analyze useful images from the ODI camera without having intimate knowledge of the ODI hardware and software details.

The project will require upgrades to the WIYN Control Room at Kitt Peak. These will include providing the necessary rack space, power, and cooling for the computers, on-site file storage, and network equipment. The requirements will be developed with the WIYN site engineer. Thirty-

two fiber optic cables, plus spares, capable of One Gigabit Ethernet, will need to be run from the ODI camera to the Control Room.

The project software development will follow an Agile model, an evolutionary approach where basic functionalities are created and then improved in an iterative manner. The method relies on a high level of collaboration among the software developers and the customer (in this case the Project Scientist) who is on site with the team. The agile method incorporates an approach that allows for the changing needs of the customer through constant evolution of a working software system.

The ODI Tier 0/1 Software Project is planned around the use of three highly qualified software developers plus the Project Scientist and the Project Manager. The ODI Project Manager will manage the project. The project leader is the ODI Software System Engineer, Andrey Yeatts who is responsible for the overall architecture and design. He is supported by a Senior Software Engineer, who has the necessary Java, J2EE, JBoss experience, and a Software Programmer, Gene McDougall, a WIYN employee with extensive experience as an operator at the WIYN telescope. The ODI Project Scientist, Daniel Harbeck, will be the liaison between the end users and the software team and will ensure that the science requirements are being properly interpreted and implemented by the software team.

The project plan, described later, extends until January 2011 and the cost estimate to complete is \$896K, which includes \$799K of labor and \$97K of computer hardware for installation on Kitt Peak.

1.2 Project Deliverables

The ODI Tier 0/1 Software Project will deliver:

- A computer cluster and network hardware for installation at the WIYN Telescope,
- Displays for monitoring and reviewing image data,
- Software that provides the means to plan, take exposures, and store images from the ODI camera, and
- Documentation
 - Architecture Description,
 - User's Console Guide,
 - Administration and Configuration Guide, and
 - Observing Tool Guide.

1.3 Reference materials

- ODI Project Management Plan
- ODI Science Requirements Document
- ODI Requirements Management Document
- ODI Operational Concept Document
- Agile Manifesto
- Book and URL references

2. Project Organization

The ODI Tier 0/1 Software Organization Chart is shown in Figure 2-1. The software development effort is led by Andrey Yeatts with a Senior Software Engineer, and Gene McDougall providing development support. Daniel Harbeck, the Project Scientist, will provide on-site user input and evaluation to help the development team. John Cavin, the ODI Project Manager, will provide management oversight of the project. He reports to the WIYN Director who reports to the WIYN Board.

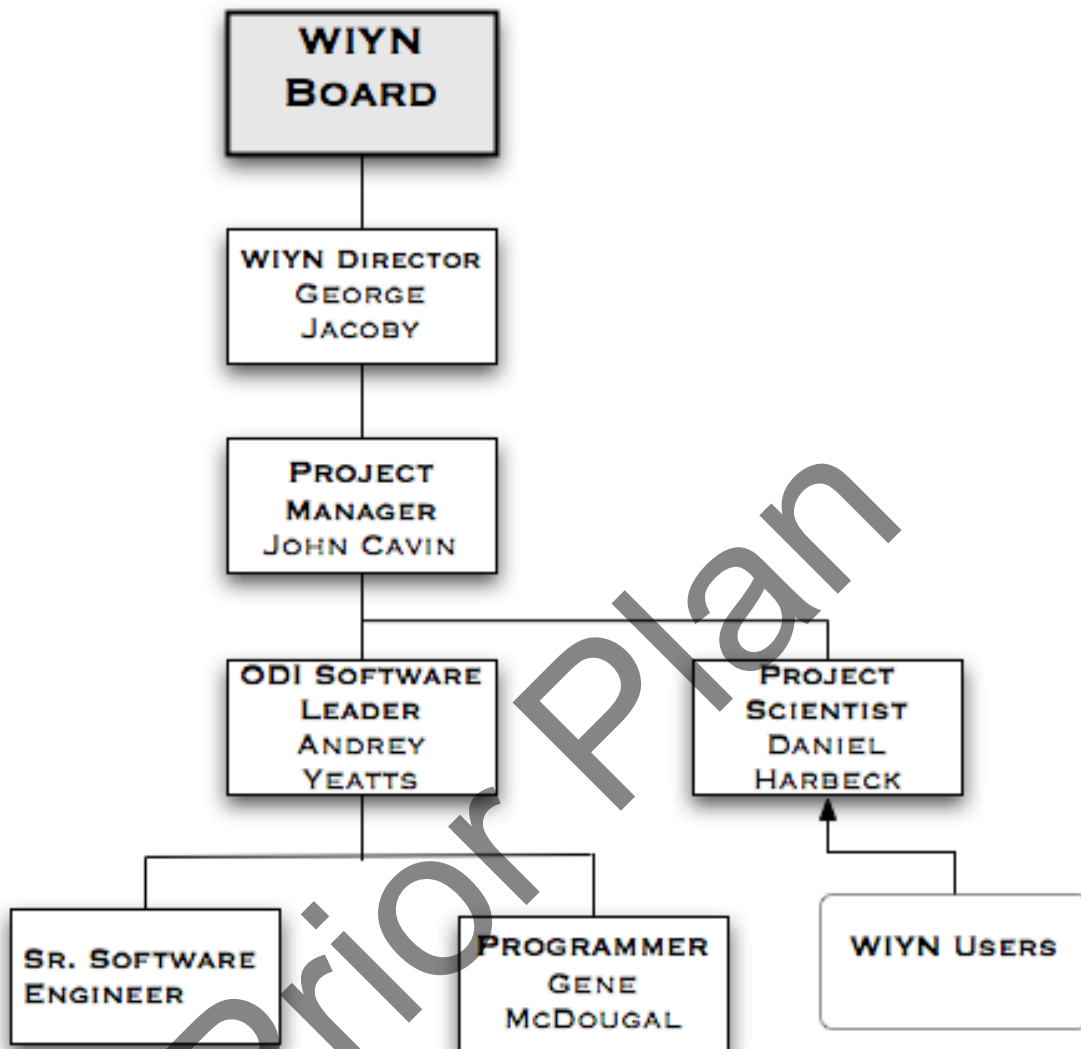


Figure 2-1. ODI Software Organization

3. Management

3.1 Management Objectives and Priorities

The objective of the Project Manager is to provide a software system that will enable the scientific community to effectively use the capabilities of the ODI camera.

ODI Software Project Priorities:

- 1) The highest priority will be to acquire images from the ODI focal plane and store them,
- 2) Control of ODI (ADCs, Shutter, and Filter Mechanism) and Telescope,
- 3) Integrated Guide Star corrections for the entire focal plane,
- 4) Quick Look quality control display of images is pseudo-real-time
- 5) Storage of raw data
- 6) Tier 1 processing (bias, dark and flat field corrections) of image data

- 7) Local archival of Tier 1 processed images with metadata
- 8) Formatting of preprocessed images
- 9) Transfer to NOAO Archive
- 10) Localized Guide Star corrections,

3.2 Assumptions, Dependencies, and Constraints

The ODI software project is based on the assumption that qualified, software professionals will be available to design and implement the software. The project has already been delayed by 1) the false start with the MONSOON controller and attempts to make it work, 2) the selection process for the OTA controller, 3) the availability of additional qualified personnel, and 4) other WIYN projects. The current plan is based on a June 1, 2008 start date with three people.

The ODI project depends on:

- 1) Skills of the ODI software personnel,
- 2) Availability of StarGrasp controllers,
- 3) Commercial hardware,
- 4) Open source software products and tools, and
- 5) Availability of user input.

The ODI software project will use commercial hardware in a Beowulf cluster configuration to reduce cost and provide flexibility for future processing requirements. The 32 StarGrasp controller nodes will be connected via fiber optic gigabit Ethernet through a network switch to the Cluster. Archival of data on Kitt Peak will be on a local RAID storage system. The cost of this hardware and its configuration is included in the ODI software project.

Several open source software products will be used in the development of ODI software. The primary programming language will be Java. JBoss, an application server that simplifies developing and deploying enterprise-level Java applications, will be used to reduce the amount of new development required and to improve reliability of the ODI software. JBoss also provides numerous other tools that support remote monitoring and execution, caching, and data persistence, i.e., data that outlives the program that created it.

The constraints of the ODI software project are time and people. To perform testing of the integrated dewar in September – December 2009 and have a well tested, functioning ODI camera for commissioning in March 2010 will require a focused effort by the software development team and user involvement in reviewing development releases.

3.3 Monitoring/Controlling Approach

The Project Manager will manage the ODI software project, monitor its progress, and report to the WIYN Director and the WIYN Board. Monitoring will be done to the work package level. The Project Manager will meet weekly with the software development team to review progress and discuss issues. They will adjust the project plan as necessary.

3.4 Staffing Plan

Andrey Yeatts is already a WIYN employee and has been analyzing the ODI software requirements, developing the ODI software architecture, and creating software prototypes and tests. Gene McDougall is also a WIYN employee and is available for the ODI software project. However, he has other responsibilities and is not available full time. In addition, he has other hours assigned to detector testing, hardware testing, operational readiness and commissioning. The staffing plan has taken this into account. The third software engineer needs to be made available as soon as possible to prevent further slips in the ODI software effort. The staffing plan is shown in Figure 6-2 is based on the tasks in the MS Project Schedule.

4. ODI Requirements

Requirements for the ODI project have been developed and documented in several documents. They are:

- ODI Science Requirements
- ODI Requirements Management Document
- ODI Operational Concepts Document (draft)

Analysis of ODI Science Requirements and their implications on ODI Software has been ongoing for almost two years, as the requirements have evolved.

4.1 Science Requirements

Science requirements for the ODI project are documented in ODI_SRD-AD-01-0008v2.2 and have been approved by the Science Advisory Committee and the WIYN Board.

4.2 Software Functional Requirements

The software requirements for ODI software are derived from the ODI Science Requirements and have been documented in a hierarchy diagram. The diagram relates the ODI Science Requirements to the Operational Requirements and the subsequent software functional requirements necessary for implementing the planned functionality of the ODI camera. The functional requirements are broken down into the six following areas: 1) What the Observer needs to prepare for observing blocks and executing scripts, 2) software functions for actual observing, 3) functions required by the telescope operator, 4) functions for controlling the HEMI, 5) functions required to program and control the 32 StarGrasp controller channels, and 6) functions required to store images in the NOAO archive. The following outline elaborates the software functional requirements.

1. Observer

- a. Pre-Observing Preparations
 - i. Prepare observing blocks
 1. Telescope position
 2. Filter select
 3. Dither pattern
 4. Imaging mode
 - a. Static
 - b. Coherent
 - c. Local
 5. Guide Star verification
 6. Pre-image for Guide Star selection
 7. ADC setup
 - ii. Create and execute scripts

2. Observing

- a. Edit observing blocks
 - i. Prepare observing blocks
 - ii. Execute stored observing blocks
 - iii. Update parameters of existing observing blocks
- b. Request sequence of integrations
 - i. Zero
 - ii. Dark
 - iii. Object
 - iv. Flats
 1. Sky
 2. Dome
 - v. Select Filters
 - vi. Select exposure time

- vii. Select number of exposures
- viii. Select/define dither pattern
- ix. Execute dither patterns
- x. Title
- xi. Imaging mode
- xii. Guide Star Setup
- xiii. ADC setup
- c. Metadata collection
 - i. Telescope
 - ii. Focal plane
 - iii. HEMI
 - iv. Generate observing log
- d. Guide Star utilities
 - i. Use predefined Guide Star in Observing block
 - ii. Find Guide Stars using short pre-image
 - iii. Guide star selection on the fly
 - iv. Manual select/deselect
 - v. Optional user approval of Guide Stars
 - vi. Re-acquire or reselect during dithers
 - vii. Low-pass telescope guiding using data from ODI Focal Plane
 - viii. Select guide mode
 - 1. Off/static (telescope guiding only)
 - 2. Off/static (no guiding)
 - 3. Local guide star guiding
 - 4. Coherent guiding
 - 5. GUI for selecting OTAs to be corrected
 - ix. Non-sidereal (option)
- e. Focus sensor
 - i. Set reference focus and sensor column
 - ii. Set exposure times
 - iii. Enable/disable auto correction
 - iv. Log focus values with filter name
 - v. Open shutter to focus
- f. Quick look
 - i. QUOTA-sized image
 - 1. Overscan corrected
 - 2. Gain corrected/flat-field
 - ii. Down-sampled or resampled full-field image to reduce bandwidth
 - 1. Overscan corrected
 - 2. Gain corrected/flat-fielded
 - 3. Navigate to display QUOTA-sized image
- g. Tier 1 Pipeline
 - i. Overscan
 - ii. Cross-talk removal
 - iii. Dark subtract
 - iv. Non-linearity
 - v. Bias subtract
 - vi. Bad pixel mask and Cosmic ray detection
 - vii. Shutter correction map
 - viii. Flat-fielding (OT corrected)
 - ix. Adjust WCS
 - x. Co-add pixel-shifted images
 - xi. Generate master calibration files (combined biases and dome flats)
- h. Post-observing
 - i. End of run cleanup
 - ii. Data storage for transport

- i. Calibrations and system health
 - i. Observing blocks for biases, flats, darks
 - ii. Analysis tools for monitoring calibration data
 - 1. Photon transfer curve
 - 2. Generate WCS
 - 3. Log, plot, flag overscan levels
- 3. Telescope Operator**
- a. Check/approve new position using video mode of central 1-4 cells
 - b. Dithering
 - c. Limits
 - i. Rotation
 - ii. Altitude/azimuth
 - d. Guider monitor
 - e. Focus monitor
 - f. Wavefront measurement and analysis using a near central cell
 - g. Focal plane temperature monitor
 - h. Dewar vacuum monitor
- 4. HEMI Control (ADCs, Filters, temperature)**
- a. Filters
 - i. Initialize all
 - 1. Populate filter table
 - ii. Select 1-3 filters
 - iii. Verify correct filter via bar code
 - b. ADC
 - i. Initialize and home
 - ii. Send to null position
 - iii. Track with telescope position and filter
 - iv. Send specified orientation for flats
 - c. Temperatures
 - i. Monitor and report
 - ii. Issue warnings
- 5. Focal Plane/Dewar**
- i. CCD Control
 - 1. Detector configuration
 - a. Set clocks and voltages
 - b. Define OTA geometry
 - c. Configure focus sensor
 - 2. Observing configuration
 - a. Define guide cells
 - i. Set special clocks
 - ii. Define guide star windows
 - iii. Dither driven window reset
 - b. Define dead cells
 - c. Set OTA mode
 - i. Static
 - ii. Telescope guide
 - iii. Local guiding
 - iv. Coherent guiding
 - v. Non-sidereal tracking
 - d. Set science integration time
 - e. Set guide star integration time
 - ii. Thermal control
 - iii. Vacuum control
 - iv. Monitor OTA controller exhaust temperature
- 6. NOAO Archive**
- a. Store raw data for one year

- b. Export images to NOAO science archive

4.3 ODI Software Architecture and Design

Analysis of the ODI Science Requirements, ODI Operations Requirements, ODI Software Functional Requirements, OTA timing and control requirements, requirements of the StarGrasp Controller, and the desire to allow for future science has led to the adoption of a modern software architecture, the Service Oriented Architecture (SOA). A mapping of the ODI Software Requirements to the Services for the six major functional areas, plus descriptions of each Service and Tools is contained in Appendix A. The following paragraph discusses the characteristics of SOA and why it matches the needs of ODI.

4.3.1 Service Oriented Architecture

To maximize reusability, enable parallel operation, and minimize module dependencies, ODI uses a Service Oriented Architecture (SOA) to build applications. The Service Oriented Architecture is a further refinement of modular programming, object oriented programming, and distributed programming that decouples the implementation of some functionality from its interface and allows access to the functionality over a network connection. Access to functionality over a network is important to ODI because of the network-based nature of the ODI hardware system and the need for parallel processing of the image data. The StarGrasp Controller provides features that handle the critical timing needs and makes the use of SOA feasible.

The distinguishing features of a Service Oriented Architecture (SOA) are:

- A group of functions are packaged into discrete units (services) with a well-defined interface. They may be considered a single remote object, or just a group of similar functions. All functions have the same lifetime; that is, they are all created and made available at the same time, and are destroyed or restarted concurrently.
- Services are distributed over a network and are location transparent. Clients use local proxy interfaces that hide the actual service location. The advantage of using proxy interfaces is that they permit relocating the services, using different remote communication protocols, or using different implementations of the services without having to modify the client's software. Proxies are obtained from a lookup, or directory, service.
- Services are addressed by a lookup service. A directory or lookup service (also called a service broker) is used to find service implementations by name or other criteria. The lifetime of the service is independent of their clients, and so they are registered with a lookup service so that clients can obtain a reference to begin using them.

JBoss or other J2EE application servers provide utilities and support for various functions. *This is software that ODI does not have to develop.* Significant features include:

- Enterprise Java Bean (EJB) containers and management
- Naming and directory services
- Transaction support
- Database connectivity
- Messaging
- Security
- Web application, Service deployment
- Remote procedure calls
- Object caching across multiple servers

EJBs are Java objects that provide services to calling programs. They are created within a container, which provides lifecycle management, transactions (all-or-none semantics for a set of operations), and an environment for locating other components of an application, such as databases, etc.

4.3.1.1 ODI Example using SOA

Applications are created by composing functions of lower level services. In the example shown in Figure 4-1, an exposure client (say a Graphical User Interface (GUI) tool or automated process) needs to run a script to perform an exposure, requiring the filter to be set, the telescope to be pointed, and the focal plane CCD controlled. The exposure service provides the interface (for example):

```
Status execute(String script);
```

The sequence of operations is:

1. The client calls the script service with its script
2. The script service looks up services for the filter, the telescope drive command system, and the filter system.
3. The filter service exports an interface routine `setFilters(String f1, String f2, String f3)`. The scripting service uses this call to change the filters.
4. The filter service obtains an interface to the WIYN instrumentation control and telemetry service. The control service provides a low level binary GWC command format used by instrumentation and telemetry services at WIYN.
5. The control service obtains an interface to a logging service to store the event of the filter change command. The GWC command is mapped to a database table row, and stored in the ODI command and telemetry database
6. When the command is completed, the filter sends a confirmation event that is likewise stored in the database.

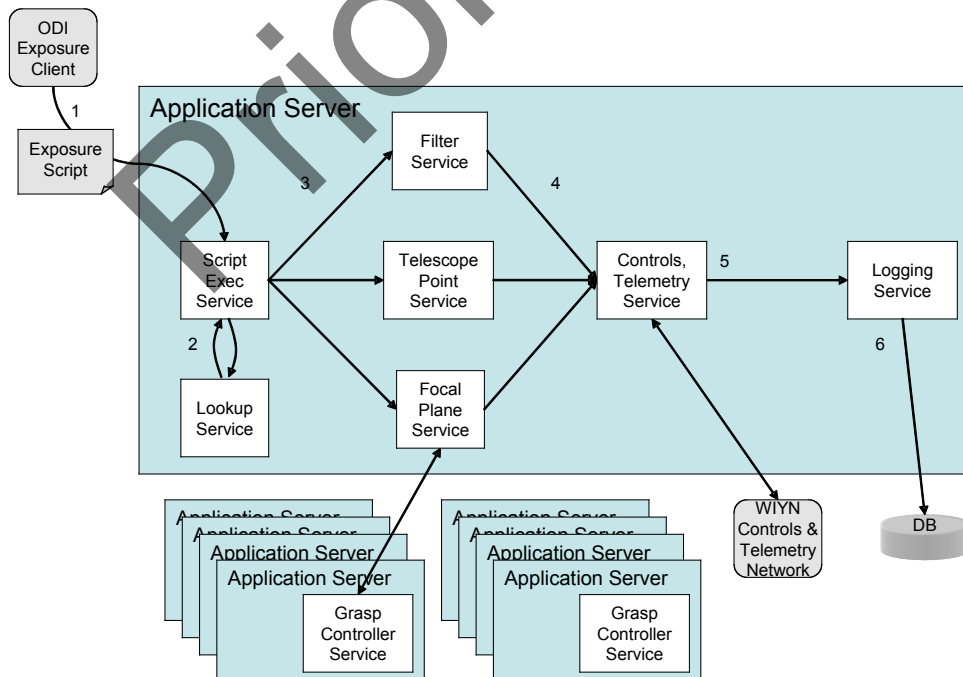


Figure 4-1. Example of ODI Exposure Client Using SOA

The use of an application server like JBoss enables the services to be packaged as “Managed Beans” (MBeans), or objects that share a common management interface. JBoss provides a web interface to start, stop, restart, create and destroy any object deployed into the application service. Once deployed, an MBean is also registered with a lookup service so that other applications may use it.

4.3.1.2 Modifying Services

By decoupling these services from their users, replacing them with engineering, debugging, or entirely new versions is simple, only requiring new service implementations to be registered with the lookup service. This feature will allow the evolution of ODI functionality as new science cases evolve without massive rewrites of the software. New applications and functionalities can be built by re-using existing services in new ways; for example remote operation or queued operations can be created without affecting the code in the services.

5. Technical Process

5.1 Methods, Tools, and Techniques

The basis of the ODI system is a distributed, clustered I/O and computation resource that handles data from the StarGrasp controllers, stores it, and processes it for Tier 0/1 image correction, quick look and archive delivery. Each exposure causes the 32 controllers to stream image data in parallel at a high data rate. Such large scale parallel systems typically use enterprise application servers that allow the application to be assembled from smaller services that inter-communicate. Each service performs a distinct, well defined operation on behalf of the larger system. This enables higher reliability and much greater adaptability to evolving requirements, including application performance scaling. We have chosen JBoss as the application server and Java 1.6 as our target implementation language.

The development method is to provide a composable service for each major component and use scripting and configuration management to produce a highly configurable system to meet ODI requirements. (A highly composable system provides recombinant components that can be selected and assembled in various combinations to satisfy specific user requirements.) The services to be implemented include:

- Focal plane camera control
- Dome and telescope control
- Instrument (filters, monitors) control
- Tier 1 Pipeline
- NOAO Science Archive interface
- Quick look GUI system
- Image cataloging and retrieval
- Telemetry management and reporting
- Scripting service
- Operator interface
- Observer interface
- Database interface

These services are incrementally developed and configured in order to meet the functional requirements of the hardware milestones, namely:

- Instrument controls integration and test (ADC, filter, shutter, temperature, vacuum, etc.)
- CCD controller integration
- Focal Plane (OTA) integration
- Instrument package pre-deployment integration

- Deployment and engineering checkout on Kitt Peak
- Commissioning

5.1.1 The Agile Process

Because of the limited budget, short timeline, and evolutionary nature of scientific processes, we will use Agile development practices. The Agile process brings several advantages to ODI since it relies on:

- Frequent, short term releases to build functionality, This means that releases can be reviewed in small increments reducing the risk of producing software that does not meet the needs of the users.
- Unit testing and test suites to test and verify functionality - Creating unit tests prior to developing the code provides a check on the understanding of the requirements for this unit of code and allows regression testing of the code after changes are made.
- Close customer involvement in the development and release review process - This means that the software is not developed in “silos” with the results tossed over the wall. The end user is an integral part of the development process.
- Ready adaptation to requirements changes or updates - Since the software is developed in short time periods, there is time to make changes to the requirements without having to make major rewrites
- Constant code evolution (refactoring) using testing to simplify and improve the code base – Since the code is developed in iterations, code is constantly being reviewed and improved.

5.1.2 Test Driven Development

One of the cornerstones of Agile development is test-driven development. This means that functional unit-level tests are put in place to ensure that the functional component is operational, even before functional code is written. Unit tests provide an automation framework to verify functionality. This supports another important part of Agile development: refactoring, or code modification to that simplifies and improves the design of the system under test.

5.1.3 Incremental releases and reviews

The development process is centered on multiple, small releases, at intervals of a few weeks to a few months. Small release iterations reduce risk and provide opportunity for guidance and course correction by the project scientist at early stages in development.

We will also seek peer reviews at intervals of the project. In addition to a critical design review, we will have local reviews of the project for the instrument telemetry and control system, pipeline, and the database model. Other milestones will be scheduled for operator and observer interfaces to verify workflow and functionality. These will be in multiple small releases with reviews and feedback from operators and observers. One such review has already occurred with stakeholders regarding the telemetry and operations database design.

This requires participation in the development process by 1) the project scientist, as the representative of the observers, 2) by one or more of the operators, as representatives of the operators who will be managing ODI on the mountain, and 3) by the telescope support staff. These customer representatives are an integral part of the team, reviewing development and guiding incremental functionality of the iterative releases.

5.1.4 Tools

Nearly all of our ODI development tools are open source. Code is developed with the Eclipse or NetBeans Integrated Development Environments (IDEs). IDEs combine a number of common tools – source code editor, debugger, build automation, class and object browsers, etc. By

combining tools, developer context switching is kept to a minimum, and productivity is enhanced. Both Eclipse and NetBeans support tool plugins to support different languages and development tools, like GUI builders, debuggers, and profilers.

A crucial tool in the chain is a source code management tool. Subversion is an open source distributed source code manager that allows multiple clients access to a source code database, and provides transacted code check in and check out. Subversion manages an unlimited number of release modules and is easily incorporated into the IDE coding tools.

JUnit is a standard open source means of managing Java unit tests. Once coded to the JUnit standard, a unit test can be combined into a test suite and invoked for regression testing as a part of the build process or other automated check.

Ant is an automated build tool. It takes a specification of the libraries and source files required by a project and builds and packages the execution units needed. Targets may include execution units (binary files), package libraries, or the test results for regression testing, for example.

JProbe is a full featured commercial profiler that instruments code at runtime to provide execution profiles for memory use, execution time, and code coverage (which lines of code are executed for a given operation). The tool is robust and can test individual applications or code packaged as libraries for deployment in an application server.

All Java code is formatted using the built-in Eclipse (or NetBeans) formatter. This simplifies style and readability issues, and lessens dependence on individual developer to adhere to a particular format or style.

5.1.5 Issue tracking

We currently have an installation of Bugzilla [see references] to track issues such as bugs and enhancements. We integrate issue tracking and code management in order to manage versions by functionality and release status as follows:

- A new issue is created. Information about the bug or feature is attached to the issue, such as supporting documents, WBS numbers, etc.
- The code in which the change is to be made is tagged, and a branch is created in the SVN repository.
- The issue is worked on in the branch to avoid interference with the trunk of the code base. Other development can proceed in parallel without losing working versions, and any previous version, along with its documentation, can be obtained.

5.1.6 System Deployment and Management

The ODI computer system will be a Beowulf Linux cluster of at least 8 nodes, each with at least 5 Ethernet interfaces, one per each of four associated StarGrasp controllers, plus one for control from the cluster head. Additional nodes will support the database, storage, and additional pipelining and quick look processing.

Compiled Java class libraries are deployed to a directory on the head node that is shared across the cluster via NFS. Code deployment can be thus managed from the head node alone.

One of the most important components of the system is the JBoss application server. This essentially provides a distributed Java environment across the cluster, further enhancing parallel programming, not the least by allowing unified management of code across the cluster. The JBoss (and Java application servers in general) provide such features as:

- Web management. All services on a node can be restarted from a web interface, without any additional programming.

- Runtime deployment of code – services can be immediately re-instantiated without requiring the restart of other services. Redeploying service code automatically restarts the affected service.
- Service naming – services can acquire references (string names or runtime objects) by supplying a name for the service to a naming system (JNDI). This allows services to be composed by scripts or configuration files.
- Remote management – services can be started, stopped, or a remote method invoked on any service within the application server, from the application server's control web interface. They may also be invoked by remote procedure call by obtaining a reference (see previous) from the application server.
- Distributed communication – service objects (metadata, image data, control and configuration data) may be shared across all instances in a cluster. Deploying a new copy of code only needs to be done to a single instance of a server; JBoss will take care of replicating the change across all servers.

Together, these provide a very powerful distributed environment. Services can be managed from the head node within a single program or configuration file, and independently communicate between servers as needed.

For example, debugging services may be added for development, substituting test versions of services (like telescope pointing, or instrument control) to scaffold systems until the services are ready.

During operations, sets of services may be shut down without disturbing others; for example, during the day, the operations and maintenance control console may continue to run on the cluster, managing pipeline operations, while the focal plane controllers and other instrument controls are powered down.

The full JBoss application server is deployed on the head node, and the JBoss micro kernel (a minimal application server) is deployed on the StarGrasp/Pipeline cluster nodes. Applications are initiated on the head node, and scripts deploy StarGrasp and pipeline control processes via JBoss remote operations.

5.2 Documentation

The ODI Software Project will produce the following documentation:

- Architecture Description,
- User's Console Guide,
- Administration and Configuration Guide, and
- Observing Tool Guide.

Using an Integrated Development Environment (IDE) also provides the tools for generating documentation, JavaDOCs, and class browsers for inspecting classes hierarchies, class methods, and details about the code structure.

5.3 Configuration Management

There are multiple configuration processes to be managed under the ODI processes. Software development products are managed with the Subversion source code version system [see references]. The version control system tracks changes in groups of files, enabling subreleases to be named and tracked.

Device configurations (clocking information, bias levels) will be maintained by a relational database. The database will provide versioning per OTA, per function (e.g., guiding, readout, etc.) and by revision level.

An issue tracking system (as mentioned previously) can manage both source code and system configuration files. As new versions and services are available, they can be tested and older, test versions of the system restored as needed.

6. Work Packages, Schedule, and Budget

In order to meet the objectives of the ODI project, the ODI software development is based on meeting milestones that are necessary for the efficient, construction, integration, and test of the hardware and software. Since we are using a Service Oriented Architecture, these milestones require that the Services have an adequate level of completion to complete the milestone. These Services will be revisited several times during the course of the project as part of the Agile process. A brief description of the milestones, completion date, and the Services/functionality required to meet the milestone are listed below in Section 6.2. Due to the delays in starting the software effort, the software functionality has been phased to meet the science requirements of having a static imager available and then adding full functionality later. The time-phased functionality is described in Section 6.1.

6.1 Functionality versus Schedule

6.1.1 Functionality available at the start of commissioning:

All functions required for science commissioning of the ODI instrument as a static imager will be available per the science requirements, including:

- StarGrasp/CCD control,
- Instrument control for the ADC, filter mechanism, and shutter,
- Focus control,
- Telemetry recording and display for instrument function, including those above and also the vacuum and focal plane temperature system,
- Telemetry recording and display for telescope and dome operations,
- Facilities to load and execute a science or calibration script,
- Cataloging of calibration and science images, including metadata,
- First tier image pipeline correction of images: bias, dark and flat corrections,
- Quick look of images providing pan, zoom and contrast/brightness manipulation,
- Database recording of telemetry and exposure events,
- Operator and observer's consoles for operation and observation tasks,
- GUIs for exposure configuration,
- Telemetry and exposure log reports

6.1.2 Functionality not available at commissioning start:

Completion of the following functions and capabilities are deferred until after the completion of commissioning, per the suggestions of the project scientist, Daniel Harbeck. These facilities may be available in an experimental capacity, but will not be available as a reliable service for science commissioning operations. It is planned to complete this functionality by January 2011.

- Focal plane tip tilt-correction and targeted photometry. This includes coherent (four corner guiding), independent guiding, and targeted photometry video acquisition. Guide cells will be sampled from edges the focal plane and used to derive a tracking error to drive the telescope. This will defer work on the command loop to drive focal plane correction, and the pipeline process necessary for focal plane guide shift correction and

other support tooling. Sufficient work on the focal plane guide shifting will be done to confirm operation of the StarGrasp controllers for acceptance testing, but operations support of tip-tilt correction and image processing will be deferred.

- Observation planning tools. The tools to aid the creation of customized observation scripting, and guiding observers in planning observations will be deferred. Basic scripts will enable acquisition of calibration images for bias, dark and flat images, and for static images (minus the video areas used for telescope guidance). GUI tools will be available to configure exposure parameters, such as exposure length, filters, pointing, focus and telescope guide stars. Support for more extensive configurations for dithering and surveys will be deferred.
- Additional pipeline operations for cosmic ray rejection, dither combination, focal plane tip-tilt shift signature configuration and targeted photometry. These will be deferred until after commissioning.
- Quick look capabilities using or supporting these features will likewise be deferred.

6.2 Work Packages/Milestones

6.2.1 Concept Demo

August 2008
Working Focal Plane simulated image readout
Scripting
Simulated Instruments
Prototype of observer interface
Pipeline overscan correction

September 2008
Quick look prototype

6.2.2 CDR

October 2008
Multiscreen (2x1280x1024) quick look
Simulated guide and guide processing

6.2.3 Instrumentation ready for test

November 2008
Run significant commands for shutter, filter, ADC
Receive events from vacuum, temperature sensing subsystem
Record results in Database
Recall and display telemetry events and command execution

6.2.4 StarGrasp controllers 1st half

December 2008
Read 32 simulated OTAs from 16 controllers
(Need 4 quad Ethernet cards)
Drive voltage, clocking tests for OTAs

6.2.5 StarGrasp controllers 2nd half

February 2009
Read 64 simulated OTAs from 32 controllers

6.2.6 Device characterization and tuning

September 2009
Produce optimized guide, readout, photometry CLV sequences
Insert into database indexed by OTA id number, function, and version

6.2.7 Focal plane ready for test

October 2009
Basic pipeline procedures ready to test
Bias, dark, flat, science image exposure processes
Overscan, bias, dark, flat pipeline corrections

6.2.8 Pre-deployment test of HEMI

November 2009
All instrumentation command and event features ready
Timing in lab has required accuracy

6.2.9 Test of HEMI

December 2009
Test of exposure processes with telescope, shutter, filter, ADC
Check time measurement for accuracy

6.2.10 Integrated test

January 2010

6.2.11 Science Commissioning

Mar 2010

6.2.12 Software Updates and Commissioning

January 2011
Observation preparation tool complete
Focal Plane Control including coherent local guiding
Pipeline completion
Scripting integration with tools, GUI
Quick Look GUI enhancements for Photometry and large format display
Manuals

6.3 Resource Requirements

The ODI Software plan is based on having personnel with experience and the proper skill set. Since the software effort is already behind schedule, we must move quickly to meet the already reduced scope. Figure 6-1 shows the staffing profile for the rest of the project. Note that the two spikes in the chart are associated with two key periods in the project, the first is when the HEMI hardware is assembled for integration and testing, and the second is for final integration and testing of the complete instrument prior to shipping it to Kitt Peak for installation on the telescope. It should also be noted that both Andrey Yeatts and Gene McDougall have an additional 700 hours assigned to other integration and test tasks that are not a part of the software plan.

Software Staffing Profile

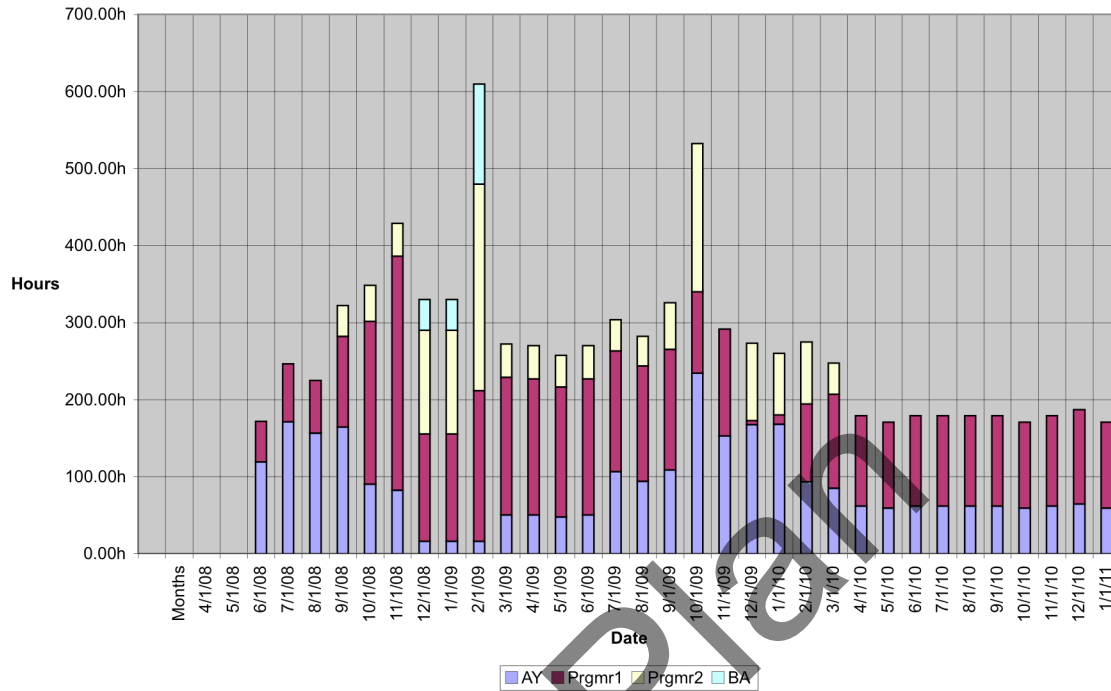


Figure 6-1. ODI Software Staffing Plan where AY = Andrey Yeatts, Prgm1 = new hire, Prgmr2 = Gene McDougall, and BA = Bahzad Abareshi

6.4 Budget

The budget for the completion of the ODI Software is shown in Figure 6-2. This includes \$799K for labor and \$97K for hardware for Kitt Peak (computer cluster, fiber optics network, and storage system). The sharp increase in cost in September 2009 reflexes the purchase of this equipment.

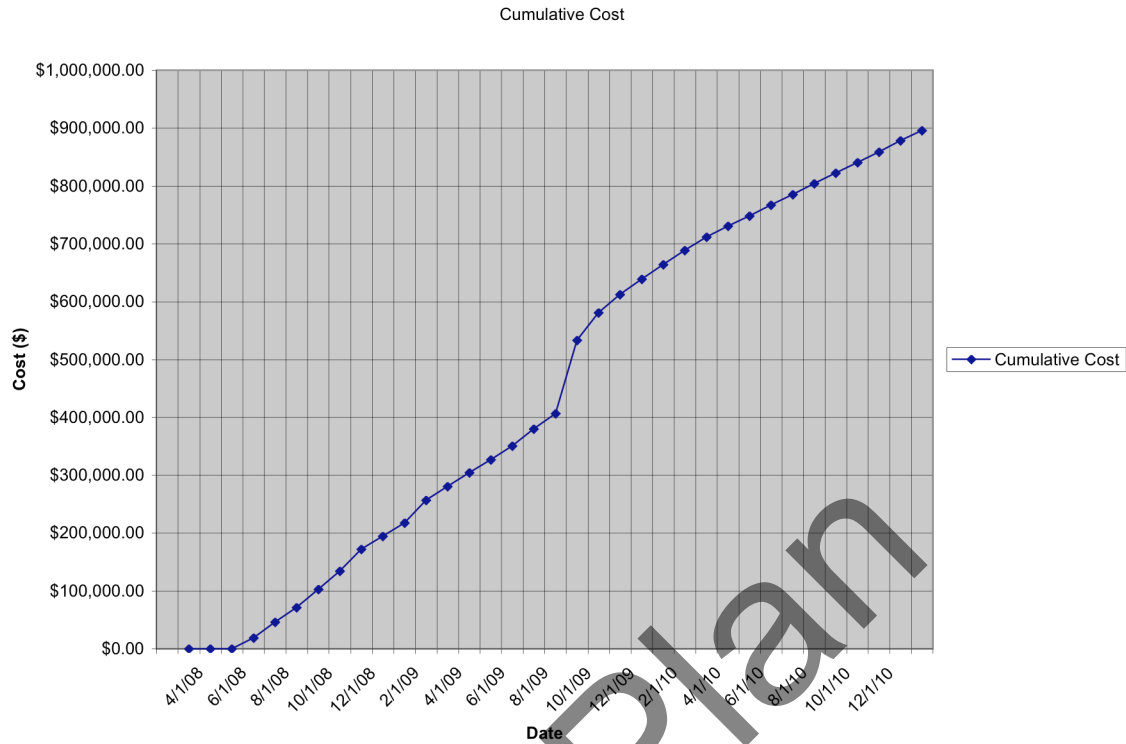


Figure 6-2. Cumulative Cost of ODI Software

6.5 Schedule

The schedule for completion of the ODI Software is shown in Figure 6-3. The Agile development process does not map conveniently to the waterfall technique used by Microsoft Project. As a result, we have mapped the software development tasks to meet overall project milestones.

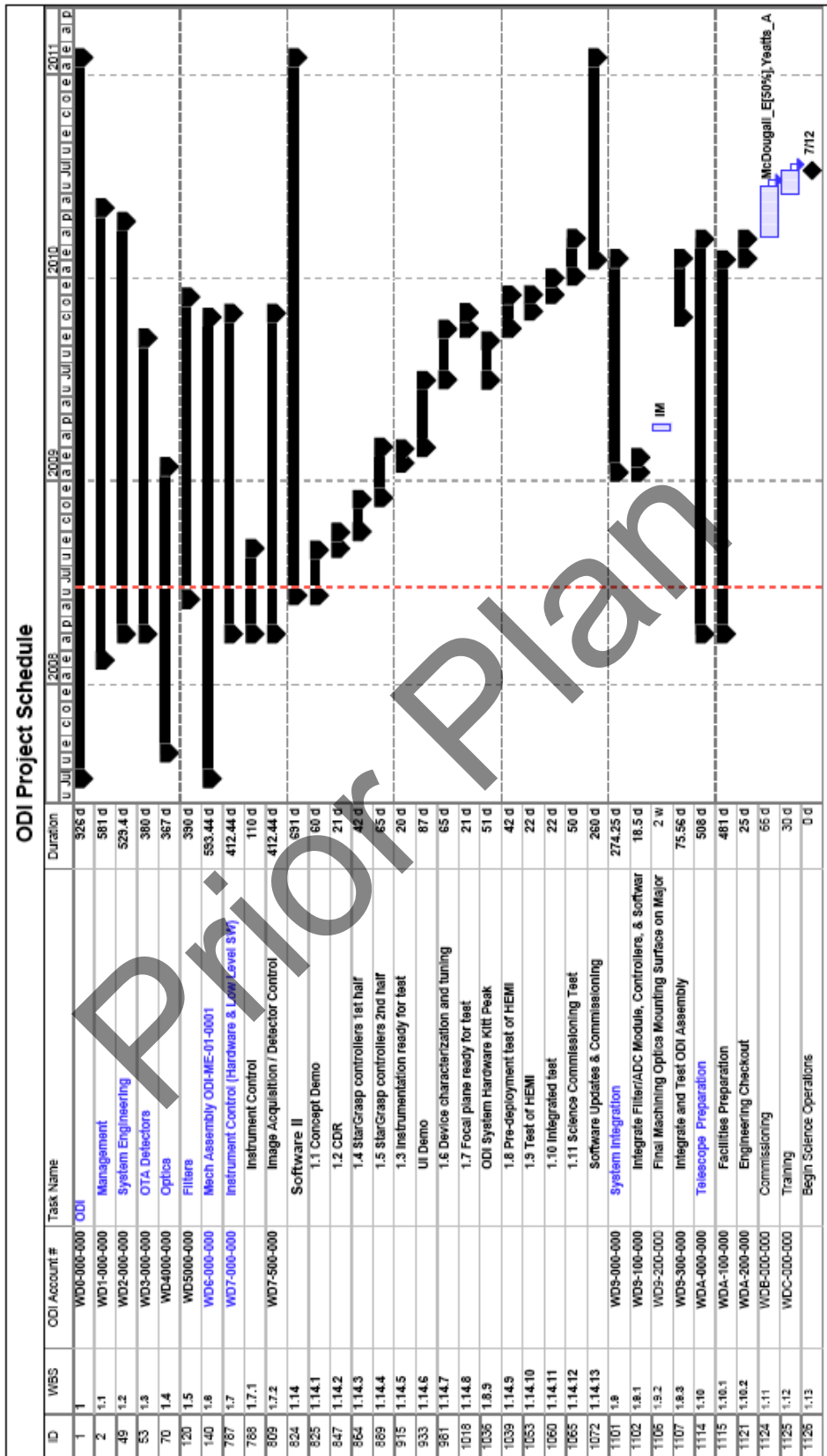


Figure 6-3. ODI Software Schedule

Appendix A

ODI Software Requirements, Deployment and Services

Since the functional requirements do not map one to one to the software components (services), it is necessary to show what software services are needed to meet the functional requirements. The following three Sections describe the mapping of ODI services to software functional requirements, where the services will be deployed on hardware, and what functionality each of the services provide.

1. ODI Functional Software Requirements Mapped to ODI Services

The ODI Software Functional Requirements are grouped into six major areas and the services required to meet those requirements are shown in the following tables. The six major areas are: 1) Observer, 2) Observing, 3) Telescope Operator, 4) HEMI, 5) Focal Plane/Dewar Control, and 6) Archive. The Service/Features are referenced by their Section number and are described in Section 3.

1.1 Observer

The following requirements list the functions that the Observer requires to be able to prepare for observations.

Service/Feature Required	Functional Requirement	Level 2 Rqts	Level 3 Rqts	Level 4 Rqts
3.5 Scripting Service 3.13 Observation Prep Tool 3.14 ODI System Simulator	Observer	pre-observing preparations	prepare observing blocks	telescope position
3.5 Scripting Service 3.13 Observation Prep Tool 3.14 ODI System Simulator	Observer	pre-observing preparations	prepare observing blocks	filter select
3.5 Scripting Service 3.13 Observation Prep Tool 3.14 ODI System Simulator	Observer	pre-observing preparations	prepare observing blocks	dither pattern
3.5 Scripting Service 3.13 Observation Prep Tool 3.14 ODI System Simulator	Observer	pre-observing preparations	prepare observing blocks	imaging mode (static, coherent, local)
	Observer	pre-observing preparations	prepare observing blocks	GS verification

3.5 Scripting Service 3.13 Observation Prep Tool 3.14 ODI System Simulator	Observer	pre-observing preparations	prepare observing blocks	pre-image for GS select
3.5 Scripting Service 3.13 Observation Prep Tool 3.14 ODI System Simulator	Observer	pre-observing preparations	prepare observing blocks	ADC setup
3.5 Scripting Service	Observer	pre-observing preparations	must be able to follow scripts	

Table 1. Software Functional Requirements for the Observer

1.2 Observing

The following table lists the software requirements necessary for observing.

Service/Feature	Functional Requirement	Level 2 Rqts	Level 3 Rqts	Level 4 Rqts
3.5, Scripting Service 3.10 Database	observing	edit observing blocks	ingest or prepare blocks	
3.5 Scripting Service 3.10 Database	observing	edit observing blocks	load previously stored blocks	
3.5 Scripting Service 3.13 Observation Prep Tool	observing	edit observing blocks	generate new blocks	
3.5 Scripting Service 3.13 Observation Prep Tool	observing	edit observing blocks	revise parameters	
3.2 Pipeline Control Scripting 3.5 Service	observing	request sequence of integrations	zero	
3.2 Pipeline Control Scripting 3.5 Service	observing	request sequence of integrations	dark	
3.2 Pipeline Control Scripting 3.5 Service	observing	request sequence of integrations	object	
3.2 Pipeline Control Scripting 3.5 Service	observing	request sequence of integrations	flat	sky
3.2 Pipeline Control Scripting 3.5 Service	observing	request sequence of integrations	flat	dome

3.4 3.5 3.12	Telescope & Inst Controls Scripting Service Observer User Interface	observing	request sequence of integrations	select filters	
3.4 3.5 3.12	Telescope & Inst Controls Scripting Service Observer User Interface	observing	request sequence of integrations	select exp time	
3.4 3.5 3.12	Telescope & Inst Controls Scripting Service Observer User Interface	observing	request sequence of integrations	select Nr exposures	
3.4 3.5 3.12	Telescope & Inst Controls Scripting Service Observer User Interface	observing	request sequence of integrations	select/define dither pattern	
3.5	Scripting Service	observing	request sequence of integrations	execute dither patterns	
3.5	Scripting Service	observing	request sequence of integrations	title	
3.4 3.2 3.12	Telescope & Inst Control Pipeline Control Observer User Interface	observing	request sequence of integrations	imaging mode	
3.11 3.12 3.5	Operator User Interface Observer User Interface Scripting Service	observing	request sequence of integrations	GS setup	
3.4 3.5	Telescope & Inst Controls Scripting Service	observing	request sequence of integrations	ADC setup	
3.8 3.10	Telemetry Mgt & Report Database	observing	metadata collection	Telescope	
3.8 3.10	Telemetry Mgt & Report Database	observing	metadata collection	Focal plane	
3.8 3.10	Telemetry Mgt & Report Database	observing	metadata collection	HEMI	
3.8	Telemetry Mgt & Report	observing	metadata collection	Generate observing log	

3.12	Observer User Interface	observing	Guide star utilities	use predefined GS in observing block	
3.12	Observer User Interface	observing	Guide star utilities	find using short pre-image	
3.12	Observer User Interface	observing	Guide star utilities	GS selection on-the-fly	
3.12	Observer User Interface	observing	Guide star utilities	manual select/deselect	
3.12	Observer User Interface	observing	Guide star utilities	optional user approval	
3.12	Observer User Interface	observing	Guide star utilities	re-acquire or re-select during dithers	
3.12	Observer User Interface	observing	Guide star utilities	low-pass telescope guider	
3.12 3.5	Observer User Interface Scripting Service	observing	Guide star utilities	select mode	off/static - tele guiding only
3.12 3.5	Observer User Interface Scripting Service	observing	Guide star utilities	select mode	off/static - no guiding
3.12 3.5	Observer User Interface Scripting Service	observing	Guide star utilities	select mode	local guide
3.12 3.5	Observer User Interface Scripting Service	observing	Guide star utilities	select mode	coherent guide
3.12 3.5	Observer User Interface Scripting Service	observing	Guide star utilities	select mode	select OTAs to be corrected (GUI)
3.12 3.5	Observer User Interface Scripting Service	observing	Guide star utilities	non-sidereal option	

3.4	Telescope & Inst Controls	observing	Focus sensor	set reference focus and sensor column	
3.4	Telescope & Inst Controls	observing	Focus sensor	set exposure times	
3.4	Telescope & Inst Controls	observing	Focus sensor	enable/disable auto correct	
3.4	Telescope & Inst Controls	observing	Focus sensor	log focus values with filter name	
3.4 3.5 3.11	Telescope & Inst Controls Scripting Service Operator User Interface	observing	Focus sensor	open shutter to focus	
3.15	Quick Look GUI	observing	quick-look	QUOTA-sized image	
3.15	Quick Look GUI	observing	quick-look	QUOTA-sized image	overscan corrected
3.15	Quick Look GUI	observing	quick-look	QUOTA-sized image	gain corrected/flat-fielded
3.15	Quick Look GUI	observing	quick-look	Down-sampled full-field image	overscan corrected
3.15	Quick Look GUI	observing	quick-look	Down-sampled full-field image	gain corrected / flat-fielded
3.15	Quick Look GUI	observing	quick-look	Down-sampled full-field image	navigate to display QUOTA-sized image
3.15 3.2	Quick Look GUI Pipeline Control	observing	quick-look	Standard evaluation functionality (e.g., MSCRED/MSCE XAM and IRAF)	
3.15 3.2	Quick Look GUI Pipeline Control	observing	quick-look	Co-add pixel-shifted images	

3.2	Pipeline Control	observing	tier 1	overscan	
		observing	tier 1	cross-talk removal	
3.2	Pipeline Control	observing	tier 1	dark subtract	
		observing	tier 1	non-linearity	
3.2	Pipeline Control	observing	tier 1	bias subtract	
3.2	Pipeline Control	observing	tier 1	bad pixel mask and CR detection	
		observing	tier 1	shutter correction map	
3.2	Pipeline Control	observing	tier 1	flat-fielding (OT corrected)	
		observing	tier 1	adjust WCS	
3.2	Pipeline Control	observing	tier 1	co-add pixel- shifted images	
3.2	Pipeline Control	observing	tier 1	generate master calibration files (combined biases, dome flats)	
3.11	Operator User Interface	observing	post- observing	end of run cleanup	
3.11	Operator User Interface	observing	post- observing	data storage for transport	

3.11 3.5 3.7	Observer User Interface Scripting Service Image Catalog& Retrieval	observing	Calibrations and health	observing blocks for biases, flats, darks	
3.11 3.5 3.7	Observer User Interface Scripting Service Image Catalog& Retrieval	observing	Calibrations and health	analysis tools for monitoring calibration data	Photon transfer curve
3.11 3.5 3.7	Observer User Interface Scripting Service Image Catalog& Retrieval	observing	Calibrations and health	analysis tools for monitoring calibration data	Generate WCS
3.11 3.5 3.7	Observer User Interface Scripting Service Image Catalog& Retrieval	observing	Calibrations and health	analysis tools for monitoring calibration data	Log, plot, flag overscan levels
		observing	tier 2, 3		

Table 2. Observing

1.3 Telescope Operator

The following table shows the requirements for the Telescope Operator and the Services required.

Service/Feature	Functional Requirement	Level 2 Rqts	Level 3 Rqts	Level 4 Rqts
3.11 Operator User Interface	Telescope Operator	check/approve new position to move to		
3.11 Operator User Interface	Telescope Operator		video mode of central 1-4 cells	
3.11 Operator User Interface	Telescope Operator	dithering		
3.11 Operator User Interface	Telescope Operator	limits	rotation	
3.11 Operator User Interface	Telescope Operator	limits	alt-az	
3.11 Operator User Interface	Telescope Operator	guider monitor		
3.11 Operator User Interface	Telescope Operator	focus monitor		
3.11 Operator User Interface	Telescope Operator	Wavefront measurement and analysis	use a near central cell	
3.11 Operator User Interface	Telescope Operator	Focal plane temperature monitor		

3.11	Operator User Interface	Telescope Operator	Dewar vacuum monitor		
------	-------------------------	--------------------	----------------------	--	--

Table 3. Telescope Operator Requirements and Services

1.4 HEMI

Service/Feature	Functional Requirement	Level 2 Rqts	Level 3 Rqts	Level 4 Rqts
3.4 Telescope & Inst Controls 3.11 Operator User Interface 3.5 Scripting Service	HEMI	Filters	initialize all	
3.4, Telescope & Inst Controls 3.11, Operator User Interface 3.5 Scripting Service	HEMI	Filters	initialize all	populate filter table
3.4, Telescope & Inst Controls 3.11, Operator User Interface 3.5 Scripting Service	HEMI	Filters	select 1-3 filters	
3.4, Telescope & Inst Controls 3.11, Operator User Interface 3.5 Scripting Service	HEMI	Filters	verify correct filter via bar code	
3.4, Telescope & Inst Controls 3.11, Operator User Interface 3.5 Scripting Service	HEMI	ADC	initialize and home	
3.4, Telescope & Inst Controls 3.11, Operator User Interface 3.5 Scripting Service	HEMI	ADC	send to null position	
3.4, Telescope & Inst Controls 3.11, Operator User Interface 3.5 Scripting Service	HEMI	ADC	track with telescope position and filter	
3.4, Telescope & Inst Controls 3.11, Operator User Interface 3.5 Scripting Service	HEMI	ADC	Send to specific orientation for flats	
3.4, Telescope & Inst Controls 3.11, Operator User Interface 3.5 Scripting Service	HEMI	Temperatures	monitor and report	
3.4, Telescope & Inst Controls 3.11, Operator User Interface 3.5 Scripting Service	HEMI	Temperatures	issue alarms	

Table 4. HEMI Requirements to Services

1.5 Focal Plane/Dewar

Service/Feature	Functional Requirement	Level 2 Rqts	Level 3 Rqts	Level 4 Rqts
3.1, FocalPlane/camera control 3.5 Scripting Service	Focal Plane / Dewar	CCD Control	detector configuration	set clocks and voltages
3.1, FocalPlane/camera control 3.5 Scripting Service	Focal Plane / Dewar	CCD Control	detector configuration	define OTA geometry
3.1, FocalPlane/camera control 3.5 Scripting Service	Focal Plane / Dewar	CCD Control	detector configuration	configure focus sensor
3.1, FocalPlane/camera control 3.5 Scripting Service	Focal Plane / Dewar	CCD Control	observing configuration	define guide cells
3.1, FocalPlane/camera control 3.5 Scripting Service	Focal Plane / Dewar	CCD Control	observing configuration	define dead cells
3.1, FocalPlane/camera control 3.5 Scripting Service	Focal Plane / Dewar	CCD Control	observing configuration	set OTA mode
3.1, FocalPlane/camera control 3.5 Scripting Service	Focal Plane / Dewar	CCD Control	observing configuration	set science integration time
3.1, FocalPlane/camera control 3.5 Scripting Service	Focal Plane / Dewar	CCD Control	observing configuration	set guide star integration time
3.4 Telescope & Inst Controls	Focal Plane / Dewar	Thermal control		
3.4 Telescope & Inst Controls	Focal Plane / Dewar	Vacuum control		
3.1, FocalPlane/camera control 3.4 Telescope & Inst Controls	Focal Plane / Dewar	monitor controller heat ventilation		

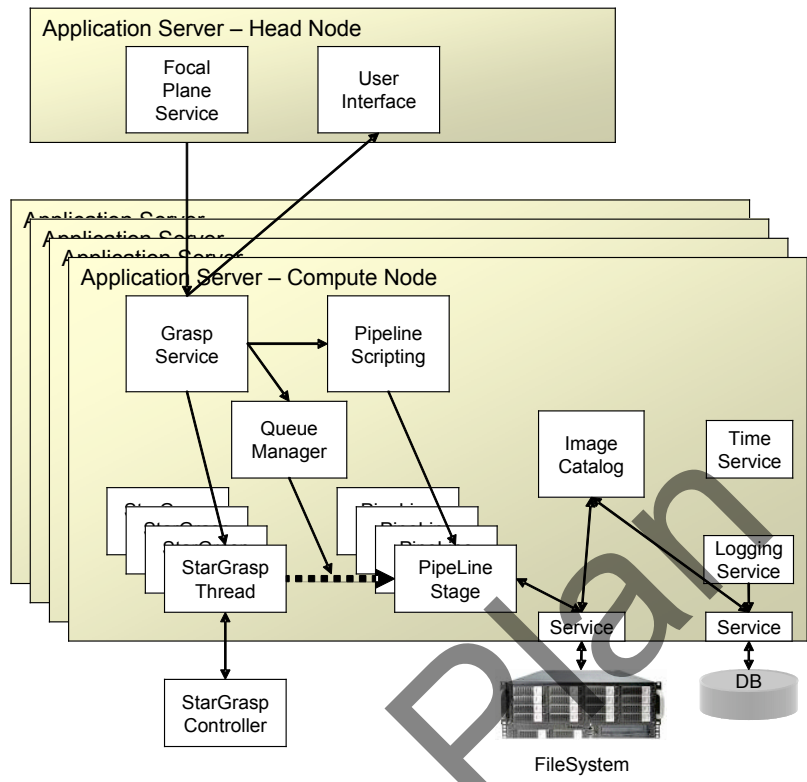
Table 5. Focal Plane/Dewar Control

1.6 Archive

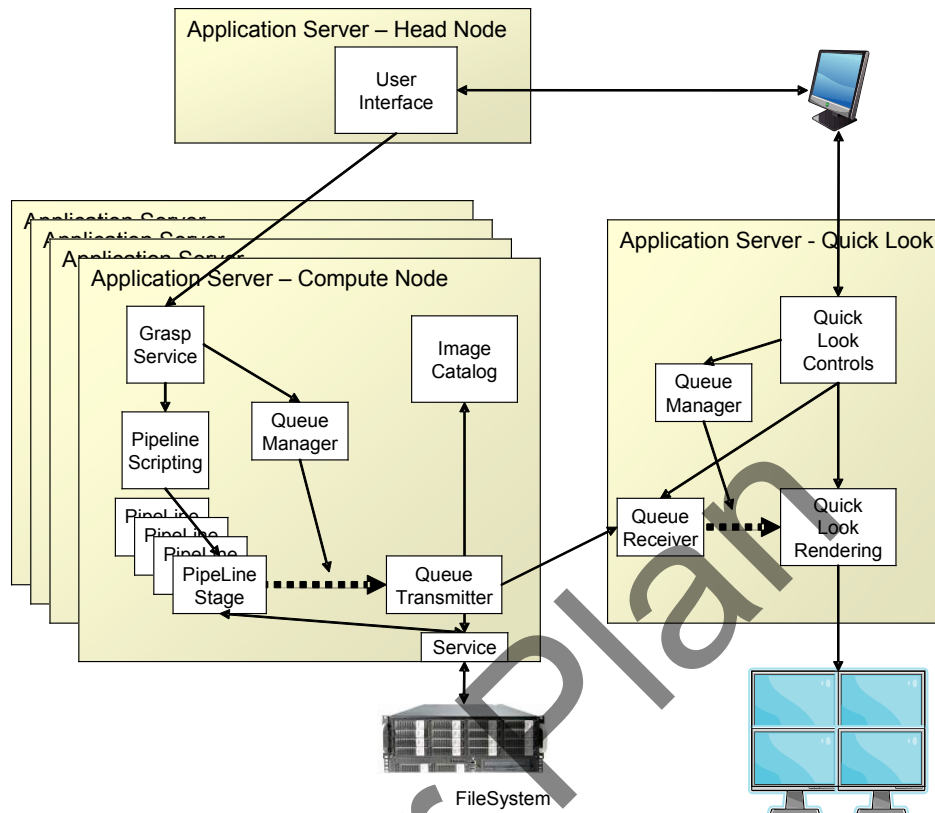
Service/Feature	Functional Requirement	Level 2 Rqts	Level 3 Rqts	Level 4 Rqts
3.6, NOAO Archive Service 3.7 Image catalog/retrieval	Archive	local raw data - 1 year		
3.6, NOAO Archive Service 3.7 Image catalog/retrieval	Archive	export to NOAO science archive		

Table 6. Archive Services

2.2 Compute Nodes



2.3 Quick Look System



3. Services and Tools

The ODI Software will be implemented by the following Services in conjunction with the software infrastructure implemented by the J2EE and JBoss enterprise software.

3.1 Focal plane/camera control

This is used to control the array of 32 StarGrasp Controllers. It consists of libraries and services on the head node of the cluster server, and server applications on each of 8 compute nodes. Focal plane and pipeline operations are integrated using the same scripting mechanism.

The focal plane control system is broken into two parts: a FocalPlane interface for user level programming and an array of GraspServer services used exclusively by the focal plane. The FocalPlane interface takes lists of StarGrasp and pipeline operations and distributes them across multiple GraspServers as lists of individual StarGrasp or pipeline operations. GraspServers accept these command lists and run each command list in its own thread for a StarGrasp, or schedule appropriately in the case of the pipeline.

The focal plane object should be configurable by

- OTA characteristics and configuration
- StarGrasp array network information
- Focal plane command server information

Each of these configurations is separable and may vary independently.

The focal plane description must contain at least the OTA configuration. This description will be available to the pipeline when processing a focal plane exposure. When further configured with

the command server and StarGrasp array information, command interface functions are available.

Typical interface functions include:

```
PixelArea getPixelArea(position, size)
setWCS(WCSFunction)
getMicronOffset(position, position)
getRA(position)
getDEC(position)
```

When network command is enabled by further configuration, the interface includes functions like:

```
HashMap<String, String> execute(GraspScript gs)
HashMap<hostString, resultString>execute(List<Command>)
HashMap<hostString, resultString>execute(HashMap<String,List<Command>>)
```

Datasources for translation (to look up clocking and bias values) are supplied before execution:

```
setTranslationDatasource(dataSourceReference)
```

Current status:

Basic functionality is available for image readout on simulator. Multiple StarGrasp controllers can be commanded from a single script through one GraspServer service. Command translation and distribution is done for single GraspServer.

To be done

- Another layer of command translation is required to fan scripts out to multiple GraspServers,
- Additional StarGrasp command implementations are needed, in particular guiding, shifting and clocking and bias configuration commands, binning configuration details
- Simulation of image and guide read for testing and support of other services
- Guide video format is yet to be determined
- Start up and shut down procedures
- Testing and debugging on real hardware
- Time synchronization

3.2 Pipeline control

The pipeline is a configurable and scriptable resource much as the focal plane amplifiers are. The same remote command server manages both.

Input images are sourced by the FocalPlane interface or by an existing exposure in the form of a set of files named by an exposure.

Each image source implies a parallel pipeline operation. Each image source generates a queue of image segments described by a FocalPlane object and other metadata. Each queue will be scheduled by a fixed thread/processor per OTA for basic tier 1 operations. Other schedules may be created for other pipeline processes (such as dither combination).

Typical pipeline operations include:

```
Launch(Queue Name, PipelineOperation po)
```

Current status:

- Pipeline can be configured to write out image file.

- Dispatched from GraspServer command

To be done:

- Tier 1 pipeline operations
- Needs to be separated from StarGrasp dispatch - dispatch in parallel (include start/wait semantics) with StarGrasp operations – expose operation currently blocks on completion of pipeline
- Metadata transformations for NOAO archive output and engineering outputs
- Configuration handling
- Pipeline status service interface

3.3 Queue Manager

The queue manager is head and compute node service used to manage (create, destroy, and locate) image queues. Each queue is named with a string, and holds a sequence of image data buffers labeled with their origin and span in the overall image. Metadata, such as the focal plane configuration or FITS metadata is associated with each image queue.

Typical interface operations are:

```
Queue createQueue(String qname, geometry information, FocalPlane fp)
List<String> getQueues()
```

While the queues are local to each compute node, the queue name space is global. That is, the names are visible to all nodes, while the data is not. Queue names are replicated across all queue manager name services. This enables operator viewing of queue status and Quick Look queue transmission requests.

Current Status

- Queue manager object exists, not bundled as a service.
- Queues can be created, looked up and destroyed by name.
- FocalPlane objects can be associated with a queue.

To do:

- Package as service
- Distribute queue names across instances
- Add FITS metadata to files read in to queues

3.4 Telescope and Instrument controls

The telescope and ODI instrument devices are accessed via the WIYN and MPG routers. A high level command interface (setFilters(filter1, filter2, filter3), e.g.) will be implemented with a network call to the router.

Typical interface functions include:

```
Filter.setFilters(FilterID filterPos1, FilterID filterPos2, FilterID filterPos3)
```

All telescope and instrument controls are forwarded to the router interface, and logged accordingly.

While not a device in the same sense as the ADC, or filter mechanism, the Focus Controller has a well defined API (Application Programming Interface) with controls and monitors. Focus commands and data (focus or photometry measurements) will be logged to the telemetry stream.

Current status:

- Prototype interfaces are defined for telescope pointing, shutter, filter and ADC devices.
- Interfaces for prototypes are stubbed for a demonstration simulation

To do:

- Build TCL or wire protocol interface to MPG and unify with WIYN router library
- Complete command set for each device
- Test against WIYN/MPG simulations

3.5 Scripting service

The scripting service is actually two parts. The first, a service to execute scripts, is fairly simply. A BeanShell or other command/program interpreter waits for a script controlling exposures or other ODI functions and executes. Error handling, logging and telemetry, startup and shutdown as well as other operating modes should be accounted for. The script execution interface is basically:

```
execute(String script)
```

The execution context will include user identity and resources such as the telescope interface, filter, and focal plane command interfaces.

The script system design is much more complicated and impacts the design of much of the rest of the system. Scripting requires the following:

- Parameterization and provisioning of StarGrasp clocking at the image, guide and other levels
- Parameterization and provisioning of typical observer features, such as exposure time, filters, etc.,
- Serialization format – persist into databases, files, remote messages, etc.
- Editable – textual, non binary format – saves tooling costs early on
- High command throughput - should be able to control StarGrasp array at highest possible rate without too much loss in scriptability, and should not require user-level thread scripting
- Expressivity – needs to describe:
 - o Parallel execution of instrument and telescope operation for exposures, efficiency,
 - o Serial composition
 - o Typical program scripting, NP complete (recursion, variables) operations

This will enable creating single threaded, observer level operations descriptions, while enabling development of high performance scripting at a more advanced level. Some scripting operations may need to be considered as transactions, to enable error recovery and management.

Pipelining will likewise be including in the scripting process. By designing the exposure process in tandem with the image pipeline, both image production and correction may be described with the same mechanism. Metadata management is made easier and management of exposure and pipeline processes is simplified. Parallelism description will shift focus from instrument operations to pipeline operations, and from instrument execution (telescope slew, filter change, etc.) to image stream and processor utilization.

Current status:

Scripting is currently implemented as Beanshell scripts with a scripting context which is supplied with implementations of the FocalPlane, Pipeline, and Instrument interfaces (section 3.1 through 3.4). Scripts can be loaded and run against simulated instruments and real FocalPlane controls (which run against simulated StarGrasps). An exposure can run the simulated pointing, filter change, etc., in parallel, call for opening and closing the shutter, and read out an image into files from four simulated StarGrasps. The read process usually takes 10 seconds or less. The pipeline implements the image file saving and is configured by script.

To do:

- Guide scripting processes – requires StarGrasp definitions
- Interface for ObservationBlock composition
- ObservationBlock execution service
- Management of scripts in databases
- Integration with external tools, GUIs, etc. to supply guide stars and other parameters

3.6 NOAO Science Archive service

The NOAO Science Archive service will take the name of the selected exposure and begin background transmission to the NS archive. The initiation and completion of NSA image transfer will be marked with a database record. Only certain well-defined image types will be capable of being archived. A metadata transformation to accommodate NSA standards will be automatically invoked on archived image types. Image types that can be archived will include standard tier 1 image products and calibrations. Image types that will not be archived will include experimental products for which no metadata transformation exists.

Status:

- Meetings with DPP, obtained documentation

To do:

- Scheduled pipeline operation with NSA metadata transformations
- Test output with DPP, repair and repeat

3.7 Image cataloging and retrieval

Images will be saved to and read from a file system on each of the compute nodes. The service for this may be either local disk (for caching) or a network attached store (NAS/ NFS) or storage area network (SAN). Naming at the scripting level will determine where images are placed. A cross index by a generated exposure ID may be used to uniformly identify exposures

Images will be owned by a user. The Operations user will have read only images used for calibration. Each Observer user will have an area set aside for that ID. Image search and retrieval will be limited by user to protect copyright or other privileges. Elaborations on permissions may be revisited later.

Calibration images will be searchable by calibration process (bias, dark, flat, other) type, time of exposure, filter, and exposure length. A catalog of calibrations will be available through the database.

To do:

- Document naming conventions
- Document data product outputs and create pipeline output stages
 - Raw and processed Images, Video, Photometry, shift lists, etc.
- Associate archival data products with archiving script

Build datapath conventions into build and scripting processes:

- Calibration images
- Raw Images
- Processed Images
- Archival Images (may be raw and processed, too)
- Temporaries (lifespan of pipeline run)
- Other

Spec and configure storage solution

3.8 Telemetry management and reporting

The WIYN router and MPG router telemetry will be subscribed to and saved to a database log. An editable, external description will configure subscription information. The current state of all instrumentation will be available to nodes in the cluster, either as a shared object or as a query service.

Status information will be logged to a database. The database will be queried for exposure-specific data for log exports to observers. Other queries and reports will be developed as needed for science, engineering and operations purposes.

Each StarGrasp controller emits a status block at timed intervals. These will be propagated to the current status service/object and logged to the database. No effort will be made to export this information to the dome telemetry routers.

Telemetry and instrument command will be time stamped using the system time service. To the extent possible, this will be synchronized with any time information available on the router services.

The logging service saves router traffic (commands and replies) and stores it to specific database tables designed for each device. Other system, such as the focal plane will have similar command and status logging. The set of logged items includes:

- WIYN/MPG/GWC command and events
- StarGrasp statuses
- Script run and exposure records

To do:

- Design, build control for logging
- Build listener application called from telemetry event receipts
- Build listener application for StarGrasp status
- Build logging for instrument commands
- Build logging for scripting commands
- Use persistence layer to save to database

3.9 Time service

Each JVM (Java Virtual Machine) (head nodes and cluster compute nodes) has a time service as a JVM-local object call that provides a unified, distributed clock across the compute cluster. It is synchronized to within 100 microseconds across the cluster and to the StarGrasp array (for any time stamped commands and replies) to within 100 microseconds. This clock is used to timestamp operations and should be in sync with the StarGrasp array's time stamping on video or other command and data streams.

Typical interface calls:

```
long TimerService.getTimestamp();
long TimerService.getTimeStampFromStarGrasp(long stargraspTimeToken)
```

Should have some estimate measurement error, perhaps by client location. StarGrasp or other translations to global time should be invertible.

3.10 Database

Several data sources need to be made available for observer and operations use. Some of these sources may need to be transportable for observation planning. For example, a guide star extract for a particular region may be necessary for remote preparation. Other databases may be translated to a different format: telemetry logs to a text event trace, for example.

Each database will be served under a separate database schema and user. References will be available in the node's JNDI service.

Guide star catalog – GUIDESTARS

- 1-2 tables
- needs subsetting process for observation prep

Stargrasp clock and bias configurations – STARGRASP

- 3+ tables – clock clv, bias, join, management

Calibration and science image catalog – IMAGES

- 5-6 tables estimate

Exposure libraries, script queues and run logs, user info – EXPOSURESCRIPT

- unk, 20 tables estimate

Telemetry recording – TELEMETRY

- 30-40 tables

Image archive processing – ARCHIVE

- 2-3 tables

Joins between the schemas will be described. Additional databases will follow convention. Each data source should have a Java persistence layer and basic documentation for create, update and delete operations.

Current status:

Designed database schema for TELEMETRY and part of STARGRASP
Running in Oracle instance

To do:

Design Java persistence layer for TELEMETRY
Prefer reverse engineering of schema to objects
Design JRuby ActiveRecord reverse engineering of schema into ActiveRecord
Design other schemas
Reuse processes to reverse engineer other schemas
Set up schema, database locations in ODI build process

3.11 Operator user interface

The operator interface provides instrument control and most observer functionality. All management and operation of ODI can be

- View image catalog (web)
- View telemetry incl. focal plane controller (web)
 - o Current exposure

- Historical
- Test instrument, focal plane controller (applet)
- View script library – test, calibration, observers, etc.
- Select and run script (web)
- GUI Applet launch for
 - Guide stars
 - Instrument status (Tele, shutter, filter, vacuum, temperature, etc.)
 - Cluster/pipeline status
- View image – applet, quick look
- Wavefront sampling – script – produces FITS in appropriate format for AO analysis
- Provides telescope pointing interlock service from observer scripts

3.12 Observer user interface

The observer interface provides ODI controls and image management for the observer at WIYN. The major functions of the interface are:

- View telemetry (web), current and historical
- View script library: test, calibration, my user scripts
- Run script
- View image, Quick look control, Guide star configuration applet
- Launch observation prep tool

The guide star configuration UI should provide the following capabilities:

- use a predefined guide star list
- find using short pre-image, suggestions from SExtractor
- manual select/deselect
- optional user approval
- re-acquire or re-select during dithers
- guide star selection on-the-fly via SExtractor or other pipeline operation
- low-pass telescope guider

The Observer interface will need limitations on search capability to prohibit examination of data that may be under copyright or other non-disclosure.

Operator and observer interface functions may migrate. It is expected that they will be versions of the same application, with user identity determining functionality. Views may be functionally targeted by selective assembly of components.

Multiple observer/operator interfaces may be in operation; however, one will have exclusive use of the instrument. The pipeline may be restricted as well. The operator can take control of the instrument back from the observer at any time.

The observer interfaces provides a generic user interface to manage:

- Security model – operator, users
- User management – create, update, deactivate, password maintenance
- Page layout
- Prototype & review – operators, observers

3.13 Observation Preparation Tool

The Observation Preparation tool allows composition of exposure patterns with some degree of automated assistance. Exposure patterns include:

- Test and engineering procedures – StarGrasp clocking or biasing tests, instrumentation checks, observer or operator health checks, etc.
- Operation patterns – cleaning focal plane CCD
- Exposure processes – readout, pre-image, etc.

As noted above in scripting, the observing tool should aid observation planning by

- Look up of common or default scripts in a script library,
- Check for syntax, mandatory parameters, parameter passing, etc.
- Provide a simple mechanism to set common observation parameters such as right ascension, declination, filter, and exposure time,
- Provide a mechanism to allow modification of less common parameters such as clocking patterns,
- Provide for composition of operations in parallel for machines that can operate in parallel (examples are telescope, filter drive, shutter and focal plane), and
- Integration with data resources for calibration, etc.

Image pipeline operations are likewise scripted. Parameterization, library usage, and composition (to the degree required by tier 1 pipelining and exposure preparation) will be available for pipeline preparation. Tier 1 primitives and scripts will be available, and documentation and code examples will be produced for user pluggable pipeline routines.

The tool will be text based initially. Later versions will have GUI or forms based guidance.

The tool will parse and check scripts for correctness. Error outputs specifying script lines in error will be produced. Execution time estimates may be produced later to support observation planning.

Observation prep will be as similar as practical to the observer interface. It will share interfaces with the observer interface, but based in a Java application. It will be possible to host the Observation prep tool at WIYN or NOAO, or run a standalone, limited functionality prep tool on a personal machine.

3.14 ODI system simulator

This is a simulator for system test and script development. It accepts an ODI operation script (exposure, etc), and provides a simulation of operation, usually testing or measuring some aspect of the script or system while simulating devices and hardware based services.

While it is not a direct product for a user, it supports a number of tools. The observation prep tool, system testing, script test and development, and queued support (not a science requirement) require this. Because scripts may not be statically analyzable for timing or even for correctness, a simple simulator can examine most run paths in a script and give a reasonable indication of live performance and runtime correctness.

Most systems can be simply emulated, for example filters, telescope position or shutter. Delays can be estimated and instrument behavior simulated by thread delays in most cases. The StarGrasp array can be simulated using the C based simulator. Java based simulation for a limited set of operations may be possible.

3.15 Quick look GUI

The quick look system should receive image data directly from the pipeline cluster. The GUI system will be hosted on a machine separate from the cluster and head node. The host for the QL will require additional graphics resources to drive high resolution, large format displays quickly. A 10Gb/sec Ethernet or Infiniband data connection will be used to drive image data to the QL machine.

Quick look will be configured for dedicated display hardware, but may be launched and/or controlled from any operator or observer interface instance.

Quick look will perform displays of the focal plane and sub-apertures of the focal plane as quickly as the hardware will allow. Recommended views are:

- Full focal plane on full screen
- OTA on full screen
- Sampled cells across the focal plane (one cell per OTA) on full screen.

Other image display includes:

- Targeted photometry monitoring
- Guide/photometry target selection

Image manipulation should include:

- Panning in any direction at current resolution
- Changing resolution to zoom in and out
- Contrast stretching and brightness enhancement

Other sub-views or helper windows should include:

- Current focal plane position (legend or map)
- Current target locations (over catalog thumbnail)
- WCS information for current views

The interface should be parameterizable by WCS and FocalPlane object. That is, display spacing of pixels, cells or OTAs should be sensitive to device configuration (feature distances), pixel offsets, display or pixel aspect ratio, etc. Binning should be handled appropriately.

The interface will consist of user graphical update commands (pan, stretch, etc.) and non-graphical commands such as:

```
QuickLook.Transfer(QueueLocator remoteQueue, String localName)
QuickLook.Display(String queueName, Params how)
QuickLook.SetDisplayParams(Params)
```

The quick look display should be capable of being launched from the observer or operator interface.

To do:

- Prototype on image queues
- Multiple display buffer on 2x1280x1024 displays
- Photometry view
- Retransmit image queue to quick look machine (Transfer command)
- Command interface, helper windows
- Launch from Observer interface
- Pan, zoom (re-sampling), brightness, contrast control
- Large format display

Appendix B – The Agile Manifesto

Manifesto for Agile Software Development

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more.

Kent Beck	Jim Highsmith	Robert C. Martin
Mike Beedle	Andrew Hunt	Steve Mellor
Arie van Bennekum	Ron Jeffries	Ken Schwaber
Alistair Cockburn	Jon Kern	Jeff Sutherland
Ward Cunningham	Brian Marick	Dave Thomas
Martin Fowler	James Grenning	

Appendix C – References

Cockburn, Alistair, *Agile Software Development*, Addison-Wesley Professional, 2001

Burke, Bill, and Monson-Haefel, Richard, *Enterprise JavaBeans 3.0*, O'Reilly Media, Inc. 2006

Crawford, William, and Kaplan, Jonathan, *J2EE Design Patterns*, O'Reilly Media, Inc. 2003

Schmidt, Douglas, Stahl, Michael, Rohnert, Hans, and Buschmann, Frank, *Pattern-Oriented Software Architecture, Vol. 2 Patterns for Concurrent and Networked Objects*, John Wiley and Sons, 2000

Loy, Marc, *Java Swing*, O'Reilly Media, Inc. 2003

Bini, Ola, *Practical JRuby on Rails*, Apress, 2007

Tate, Bruce, and Hibbs, Curt, *Ruby on Rails: Up and Running*, O'Reilly Media, Inc. 2006

"Service-oriented Architecture," June 2008, http://en.wikipedia.org/wiki/Service-oriented_architecture

McGovern, James, Tyagi, Sameer, Stevens, Michael, and Mathew, Sunil, *Java Web Services Architecture*, Morgan Kaufman Publishers, 2003

"Subversion Version Control System", 2008, <http://subversion.tigris.org>

"Subversion (software)", [http://en.wikipedia.org/wiki/Subversion_\(software\)](http://en.wikipedia.org/wiki/Subversion_(software))

"Bugzilla Issue Tracking", <http://www.bugzilla.org>