GMT software and controls overview
José M. Filgueira, Matthieu Bec, José Soto, Ning Liu, Chien Y. Peng
GMT Corporation, 251 South Lake Ave., Suite 300, Pasadena, CA 91101 USA

ABSTRACT

The Giant Magellan Telescope Organization is designing and building a ground-based 25-meter extremely large telescope. This project represents a significant increase in complexity and performance requirements over 8-10 meter class telescope control systems. This paper presents how recent software and hardware technologies and the lessons learned from the previous generation of large telescopes can help to address some of these challenges. We illustrate our model-centric approach to capture all the functionalities and workflows of the observatory subsystems, and discuss its benefits for implementing and documenting the software and control systems. The same modeling approach is also used to capture and facilitate the development process.

Keywords: GMT, control software, observatory software, modeling, DSL

1. INTRODUCTION

The Giant Magellan Telescope (GMT) is a ground-based 25-meter extremely large telescope (ELT) [1]. The system level Preliminary Design Review will be held in Q1 2013. Several activities in software and controls are being carried out in support of the design review: the design of a baseline architecture, the detailed specification of the system, prototypes to demonstrate the feasibility of the architecture, and metrics to inform the development plan and budget.

The GMT system design presents a significant increase in complexity and performance requirements over 8-10 meter class systems. The GMT design includes an integrated adaptive optics system with several modes of operation [2]. In addition to the technical challenges, different partners on the project will also develop the software. The design goal is for the system to work coherently as a single unit. Careful design of the user interaction and workflows will also be necessary to guarantee the successful and efficient operation of the GMT systems.

The combination of adaptive optics modes, the suite of instruments, and requirement demands on efficiency and productivity, provide new challenges to the design of the software and control systems. The inherent complexity cannot be addressed adequately without a strong focus on design starting at the ground level of the project. The long lifespan of the project, both during development, construction and operation will render some existing technologies obsolete. This will require a delicate balance between choosing technologies early enough to provide a common framework to enable distributed and organized software development among partners, yet, late enough to take advantage of new developments that may enable or simplify the implementation.

The recent years have witnessed significant number of technologies, processes, methods and tools that could be used to address the GMT software and control design challenges. Many of these new technologies are currently being used to implement software with much shorter development cycles and increasingly complex functionalities. For example, a core body of knowledge has been developed in recent years to address user experience [3], user interaction, and user interface, mainly driven by the needs of sophisticated web and tablet applications. How to apply the novel technologies to an ELT project, where software and controls are embedded in a larger and more conventional Stage-Gate project structure requires careful strategy.

The rest of the paper will present some of the technologies that GMT is envisioning to be used to address all these fascinating challenges. We start by presenting a general framework for modeling, by use of a set of Domain Specific Languages (DSL) (Section 2). The different models attempts to capture the workflow, functionalities, requirements, specifications, and features of the entire observatory, the telescope, instruments, subsystems, and software and controls. The use of DSLs provides a consistent framework for developing the architecture of the entire system (Section 3). Section 4 shows a prototype of how hardware controls is implemented using the current architecture. Lastly, we provide a model used to capture the development process used by GMTO to systematically track the progress of development and implementation of the software and controls systems (Section 5).
2. MODELING

The role of the **semantic modeling** process in GMTO is to capture the workflow, requirements, functionalities, specifications, etc. of the entire observatory, telescope, instruments, and every subsystem that needs to be controlled. The byproduct of the modeling serves both as documentation on the software and hardware development, and as direct feed into software code generation, development, and runtime use.

The last ten years have seen a significant evolution of system and software modeling technologies. These technologies are used both to capture and represent the problem space and to drive the development of a system. Each of these technologies emphasizes different aspects of the modeling activity. Some of them support general system modeling (e.g. UML, SysML, OWL ontologies), others are based in a component architecture (e.g. AUTOSAR, OMG CCM, IEC 61499), while others place more emphasis in the execution semantics or real time behavior (e.g. OMG xUML/iUML, AADL, UML/MARTE, JPL MDS State Analysis).

The control system of an ELT is fundamentally a Distributed Real-time Embedded (DRE) System. Some of the modeling techniques (e.g. AADL, OMG SysML, OMG RTC, UML/MARTE) address specifically these types of intrinsically complex systems. The AADL standard, for example, provides semantic definitions and syntax to formally describe a real-time architecture in terms of how specialized components interact with each other. The Object Management Group Model Driven Architecture (OMG MDA) approach may seem unwieldy at first, but its layered modeling approach provides a useful conceptual framework. UML could be too general and the tools are not easy to integrate into the system. Synthesizing the benefits of different modeling tools is the philosophy that we adopt in this project.

2.1 Domain Specific Languages

In an observatory, many software and hardware components share common frameworks. Often, the code required to operate components is repetitive, with minor variations. By abstracting the variability in boilerplate code, and exposing the variability to the developer through simple configuration, we can generate uniform solutions rapidly, consistently, with minimal human errors, and cost effectively. This is the sweet spot for Domain Specific Languages (DSL).

A DSL is a computer programming language of limited expressiveness focused on a particular domain [7]. A DSL facilitates productivity and communication with domain experts. At the same time the use of a DSL helps to involve domain experts in building the model and the system. A DSL can also be seen as a veneer over a model or as a front-end to a framework. In this context the framework is the semantic/domain model.

DSLs come in many flavors and have been around for a long time. Some DSLs are based on language workbenches (e.g. Xtext, Intentional, MPS) or as internal or external DSLs. The common use of modeling integrated development environments (IDE)s involves a significant lock-in and often does not provide continuity with the rest of the system, architecturally or conceptually. In the landscape of web development frameworks, small expressive DSLs are usual and many of them are built with the same tools used to implement the final applications. The cost to access and implement a DSL is small compared to the complexity of the API behind some of the Eclipse EMF modeling tools. DSLs provide many benefits in productivity and sophistication of solution as we will see later.

The long-term cost of DSLs tends to be much lower than other approaches, such as using general-purpose languages (e.g. UML) for a specific purpose, because DSLs are crafted to fit a specific domain more efficiently. UML/SysML have profiles and stereotypes that also allow one to define a domain specific representation, but they bring with them the heavy burden of OMG, whose standards sometimes have more to do with guaranteeing interoperability between commercial IDE developers than with the needs of effective modeling tools.

2.2 GMT Modeling Framework

The wide geographic distribution and the long duration of the project make it almost mandatory to use modeling tools to design and specify components and interfaces that will remain relevant in the long run. The use of DSLs for specifying control systems helps to avoid the loss of conceptual accuracy in the use-case diagrams. For embedded systems, software expected to have long service lifespan or to run on many different platforms, it will be necessary to create additional platform-specific models outside a DSL.

We are developing a modeling framework (Figure 1) based on the Meta Object Facility (MOF) layered modeling approach [4]. The metamodel is loosely based in the EMOF/ECore metamodels. The use of a layered modeling approach
allows us to validate the conformance of each model layer with the previous, higher and more abstract layer. This framework allows more precise definition of validation rules that are specific to a particular domain.

In order to facilitate the communication and discussion with domain experts, a set of DSLs is used to populate the domain model. These languages are mostly text-based although they support graphical views to facilitate analysis and navigation. In some cases they provide a tabular metaphor (interface) mimicking the use of Excel tables used frequently in engineering design. These interfaces are often used by the domain engineer to help construct and iterate on the DSL appropriate for a given domain. However, the end user or domain engineer need not know about, and is generally oblivious to, the existence of a DSL.

The domain model is represented as object literals and is executable. The DSLs are built using Coffeescript, a transpiler (i.e. source to source compiler) that compiles into JavaScript, which is inspiring some of the new features to be available in the Ecma TC39 Harmony specification [ref]. As internal DSLs they can be extended with the features of the host language and share the same clear and expressive syntax (Figure 2). When the DSL definitions are executed, the modeling framework imports the model element definitions into a MongoDB database. Modeling tools, built using Coffeescript (e.g. code generators, model analyzers), interact with the MongoDB database. Using the Javascript ecosystem allows the reuse of existing libraries, making it extremely efficient to develop a sophisticated modeling environment. These tools can be run in the browser or in the server using node.js, an event based library that runs on top of Google V8 Javascript engine. Node.js is used in the implementation of some GMT applications, leveraging once more the symbiotic benefits.

Dynamic languages like Ruby, Coffeescript, or JavaScript greatly simplify the design of internal DSLs. Moreover it is straightforward to integrate semantic models directly into the runtime system. Another set of DSLs, sharing the same principle and syntax, has been defined to populate the rest of the model framework layers (e.g. metamodel). These models are stored using the same mechanism and database, but in different namespaces. These namespaces are accessible in runtime and are used to build development tools as they provide inspection mechanism at all levels. This allows us to check model conformance with the upper layer or to efficiently develop generic tools (e.g. model element editors). The DSLs represent a seamless, concrete, textual syntax for all the model layers, allowing the use of the same toolset to manipulate all the model layers. In addition to sharing the same syntax and storage format, all the DSLs support mix-ins, parametric mix-ins as well as closures, nested closures and inheritance. A common framework, gmt.core, provides support for the definition of the languages and models (e.g. DSL Builder).

As the project progresses and the fidelity of the models improves the DSLs will blur the line between modeling and programing. In many cases what we are modeling now will be eventually part of the final implementation of the system.
The runtime configuration can be generated in many cases from the model layer. In a distributed real-time embedded system, this configuration is mostly static. A synchronized local copy of the model is available in every computing node at runtime. This avoids the need for a central repository and simplifies distributed development and testing.

In addition to storing the domain model, the semantic model database is at the core of the system definition that drives code development and documentation. The framework provides a set of tools to support code generation. Both transformer generation and template-based generation are supported. The database stores all the information necessary to build the GMT software system so that no two pieces of similar information need to be entered into the system more than once during the whole life of the project. The implications in terms of productivity are not small: development artifacts are always up-to-date. Documentation, ICDs, plans, and so on are generated on-demand using the contents of the semantic model database. Every piece of information entered into the semantic model is part of the runtime and online documentation during operations. The efforts needed to translate between development and documentation, the propagation of human errors, and the need to manually update documentation, are thus reduced to absolute minimum.
3. ARCHITECTURE

The software necessary to operate and control an observatory can be modeled and explained using a set of essential concepts that expand several domains of knowledge, e.g. observatory operation, device control, data processing, process sequencing, hardware I/O. One of the goals of the GMT modeling effort is to capture each of these domains with a dedicated specialized DSL. The rest of the section presents some of these domains and the main concepts than help to articulate the architecture.

3.1 Component architecture

A Component\(^1\) architecture represents the main building blocks of the System and the relations between them. Architecture designs tend to be more successful when components map the concepts of the problem space closely. The closer to the domain, the more expressive will be the architecture and the more straightforward will be its implementation and adaptation to changes in requirements. The software of an observatory encompasses different types of knowledge domains or ontologies. Each domain is expressed with different concepts that can be represented by different kind of components in the architecture. As presented in the Section 2, a set of small DSLs help to capture these concepts. The GMT metamodel defines the basic components as a collection of Features.

Some components are similar enough that their commonality can be captured in a software framework. During the modeling process, the commonalities between different components can be factored out to become the basis feature set of a common framework. A mechanism is provided in the GMT DSLs to factor out a common subset of component specifications and convert them into framework specifications. These specifications are re-entered in the component definition as mix-ins (e.g. all the common features of components that must perform motion control functions can be extracted and synthesized into a common motion control library).

There is another path to specify frameworks. Existing standards or common, pre-existing, solutions to similar problems can be the basis of a framework (e.g. in the same example as before, the motion control library is based in the PLCOpen Motion Control Library standard). Based on previous experience from other projects, there are recurring patterns that already give us indications about what makes for natural common specifications. Usually a framework specification will have a matching framework implementation.

3.2 Controller architecture

Controllers are the most frequent category of components in a telescope control system. Controllers like other Components are specified using a set of Features. Controller’s features are organized in a finer grain collection than that of a Component called Function Block. Function Blocks are based on the IEC-61131-3 and IEC-61499 function block concept. Function Blocks are classified in several categories: logic, control, safety, calibration, diagnosis and vertical integration. These categories are used to emphasize the variety of functions associated normally with Controllers otherwise a natural tendency is to focus mainly on the basic logic or control functionality, leaving proper design of other crucial aspects of control as and often the rest of categories are included only as an afterthought, sometimes as late as during commissioning.

The metamodel defines several classes of Features that can be used to define a component: Property, Command, Monitor, Alarm and StateMachine. These Features have simple semantics that facilitate the communication with the domain engineers and capture the observable behaviors from the outside of a component. There is no assumption about how the features will be implemented, although Controller model specifications will have often a direct translation into runtime objects in the runtime system.

Function blocks are the basic building blocks for reuse of functionality. Control related frameworks are expressed using Function Blocks. For example, in designing a control framework that implements common functionality in control applications we specify a set of function blocks for basic controller logic, linear or rotational motion and so on. When a specific controller component is defined, a framework function block can be mix-in in the specification. This approach provides a consistent way of specifying similar functionality throughout the system. Some of the function blocks definitions are compatible with existing industrial standards, so it is straightforward to delegate the implementation of

\(^1\) Italic words are also keywords in the GMT DSLs
this functionality to an off-the-shelf product in the implantation (e.g. the motion control library function block can be implemented with an embedded motion controller that implements the PLCOpen Motion Control Library).

![Diagram of Controller component and artifacts generated from model](image)

**Figure 3 – Controller component and artifacts generated from model**

Often Controller components need access to the hardware Devices (e.g. motors, sensors, see Figure 3, top). Devices, as explained in section 4.1 have profiles usually defined by a set of XML files. These profiles are based on very stable industrial standards; the profiles are converted into our DSLs and fed into the domain model (Figure 3, middle). From this moment on they become part of the GMT modeling palette. In addition, a DSL to specify the IOModule layout of a control subsystem has been developed; this model allows automated generation of the process image that can be accessed by a Controller. This provides a platform independent way to specify and implement the hardware interface (figure 3). This logic mapping between process image and Controller features is completely independent of the protocol used to access the hardware. Adapters provide independence from the specific distributed middleware [5].

While specifying Controller features, the following elements are defined as well: Devices to interface with, observatory workflows, management workflows and UI presentation requirements. In the design process, the functional description of the Controller features is refined and a hardware deployment model is defined. The hardware deployment model includes the definition of processing resources and the Input/Output modules to interface with the hardware. All this information is captured in the domain model and used to generate code skeletons and implementations.

All these concepts are being developed and prototyped in the GMT controls lab. A big effort is being made to capture the problem domain in a way that maximizes the resources availability from industry and standards.
3.3 Observatory domain architecture

Observatory automated operations are being captured with a Workflow specification DSL. Workflows are expressed as Tasks, have Roles associated with them, Resources required, Rules and Constraints. Workflows exist in a library as potential realizations (e.g. operating modes). Workflows become Activities once they are scheduled in the observatory Calendar. The calendar can be seen as a Kanban, or planning board, were the activities backlog could be managed in an Agile metaphor of an Observatory. Workflows in combination with Pipelines and Filters are used to model data processing pipelines.

Activities scheduled for execution have an associated Sequence. Sequences describe the set of events that have to be triggered in order to accomplish an observatory activity (e.g. observation execution). Although the underlying basic technology exists in previous generation telescopes, ELT represent a big jump in terms of complexity (number of elements and number of connections between them). Managing this complexity in an efficient way while keeping the system robust, reliable and understandable is one of the main challenges for the software design. It should be possible to answer at any given moment the following question: what the system is doing now. The use of integrated adaptive optics, complex operation modes and the possibility of using more than one instrument at the same time make answering this question not a trivial matter. Traditionally, sequencing has been done in a hierarchically pattern: high-level sequencers control subordinated sequencers and so on. This makes complexity manageable, with loose coupling between different subsystems/sequences. However, this approach makes it difficult to implement mechanisms to synchronize different threads of execution at different levels of the hierarchy. It also makes it difficult to visualize where every subsystem is at any given moment in time. Our approach provides a mechanism that enables independent development of parallel sequences, synchronization between subsystems and understanding of the behavior of the system at any given moment.
Sequences are defined in a two dimensional grid (Figure 4). On one dimension, the different Components that will participate in the sequence, a Track, is defined. Tracks are executed in a concurrent way. The other dimension specifies a sequence of Steps. Every step (e.g. PRESET, AO-LOCK) has a set of Goals and Conditions that have to be met before moving to the next step. In the intersection between every track and step there is a Cell. Cells are composed by a set of Commands that often have Parameters.

For example in a sequence corresponding to making a realtime science observation, the workflow is stored as a sequence template in the Activity Library. The workflow (e.g. acquire target, obtain exposure, calibrate image, etc) follow a sequence of steps, each of which has goals and conditions to be met before advancing to the next step. During the definition of the template the parameters that need to be provided by the user (e.g. astronomer, operator) are identify.

In addition to Tracks and Steps, the sequence includes Rules and Validations. Rules are conditions that have to be met during the execution of the sequence. They are different from observatory rules as they are only in place during the execution of the sequence. Validations are used to check the conformance between parameters of the sequence.

To facilitate understanding what the system is doing at each step, a graphical tool allows editing, visualizing and executing a sequence in a two-dimensional arrangement. The graphic tool allows direct access to any parameter or command of the sequence. Any modification can be stored as a snapshot and recalled later. This feature will be especially useful during commissioning. The tool also allows us to retrigger any command directly without the need to re-run or restart the sequence.

In this approach the sequence is not generated automatically from an activity (e.g. observation), as this will require the generator to know about any combination of instrument, AO, and telescope observing modes definition, instead, a browser based observation definition tool is generated specific to each sequence template, showing what is appropriate for that sequence. The tool allows the observatory staff to create and commissioning new sequence templates that address new or refined observing modes without costly rewriting of a sequence generator or editing a sequence which is often not normally human readable. Having the domain model accessible at runtime allows the user to navigate the command space of the system and to obtain information about the definition of any command, property or parameter of the system.

4. PLATFORM

The platform defines the technologies that will be used to implement our system. Not only does the technical feasibility play an important role choosing these technologies, but also long-term availability, commercial or community support or cost effectiveness plays also an important role. Fortunately, several choices are available both in the commercial and open source world to address the requirements of the GMT software and controls. This means that several platform architectures that will provide and adequate implementation are possible. A set of criteria can be applied in order to select an optimal solution, or at least one of the most adequate solutions to a given project. Some of the criteria that we are applying at GMT follow. Avoid choosing vendor lock-in solutions when several solutions based on open standards or even open source already exist. Avoid niche market solution when alternatives that address competitive markets where economy of scale plays an important role exist (e.g. industrial automation).

The GMT project office team has been evaluating different technologies as part of its preliminary design work. From mature to more emerging, but somewhat new compared to previous generation 8-10m telescopes. The study is intended to feed into a show case feasibility design study for the project preliminary design review. Areas covered in this effort targeted different technologies: open standards (firmly grounded in industrial controls), open source versus commercial packages when available, Linux real time kernel capabilities, solutions widely employed in the high performance computing community.

4.1 Hardware interface

Industrial fieldbuses have moved from serial based protocols like CAN or Profibus to Ethernet based protocols in the last years. Ethernet based protocols are a successor of the VME backplanes used in previous generation telescopes and have become one of the most common ways to connect input/output modules with processors, especially in industrial automation. These protocols are based on industrial standards, which tend to be long-lived and very stable. This makes them an ideal choice for the long-term time span of the GMT project. Among the different alternatives, we have found
that EtherCAT is the most attractive for several reasons: EtherCAT is now an ISO/IEC standard. Several implementations of its components (e.g. master and slave stacks) are available in many platforms, including open source alternatives. EtherCAT is used to connect hardware devices (e.g. motor drives, temperature sensors) to the fieldbus via input/output modules. These models implement a standard profile, most of the time, based on the CAN open device profiles (e.g. IEC 61800-7-201/301 – CiA 402), thereby guaranteeing the availability of components from different sources.

We have performed extensive testing connecting EtherCAT remote I/O modules and motion drives with real-time computers via fiber optics (Figure 5). The tests, using cycle times between 0.5 and 0.1 milliseconds (time to read inputs, process them and write outputs from the real time computer), have shown a remarkable stability. This solution allows us to locate the processing units to the electronics room instead of the telescope structure. The welcomed consequences of this arrangement are that there is less heat to remove, requires smaller space and mass in the telescope chamber, reduces cost significantly, and makes for easier maintenance.

Industrial automation devices are becoming increasingly smarter, as low-level controllers are increasingly being embedded directly in the devices, precluding the need for separate external controls. The interface protocol with these local controllers is defined by the standards profiles. These profiles are available as XML files. We have prototyped some development tools that expose these protocols to the control applications in a systematic way. Interfaces based on industrial protocols remove the burden from the PO to distribute integration frameworks.

4.2 Basic platform and services

Linux and real-time Linux have seen extended use in scientific and control software projects. The tests performed so far (RTAI, Xenomai, PREEMPT-RT) have shown that the real-time performance and stability of RT Linux is adequate for the development of the GMT control system.
IEEE 1588-2008 (Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems) specifies the Precision Time Protocol (PTP). Time accuracy is required for the correct pointing of the telescope and for the accurate and consistent generation of telemetry timestamps. Time distribution and synchronization between real-time computers and the end-to-end system latency (a few microseconds) have been tested. The tests show that PTP is adequate for the needs of the GMT control system.

A low latency, high bandwidth and scalable communication backbone is a requisite for the design of a modular adaptive and active wavefront control system. A 40 Gbps Infiniband network setup has been tested on the lab. The test includes a combination of high bandwidth middleware, Infiniband uverbs, MPI and RDMA on a mini-cluster setup. The preliminary results show that existing ultra low latency off the shelf communication products provide the required performance out of the box. More elaborate tests that model the expected control and telemetry traffic of the GMT system will follow in the next months.

These preliminary test results show that a design based on widely available industrial open standards is capable of meeting the requirements for the GMT control system. These standards ensure no vendor lock-in and long-term product life cycle availability.

In addition to individual tests to characterize the different design choices, integrated tests will ensure that the design of GMT software systems is based on solid technological grounds.

4.3 Top and mid-tier

Recent years are seeing a frantic development of web-based technologies, both on the front-end client side and in the server side. The advances focus in empowering user interaction with new APIs in the browser, or scalability and reliability in the server side. Those advances make these technologies very attractive to the development of the top and mid-tier software components of an observatory.

GMT is developing several prototypes to validate the use of these technologies. These prototypes include the use of NoSQL databases like MongoDB, scripting languages like CoffeeScript, UI libraries based on HTML5 and SVG like d3.js, event-driven environments like node.js or high performance middleware libraries like zeromq, MessagePack or ProtocolBuffers.

4.4 Development view

A layered development architecture addresses the problem of package development, dependencies and platform evolution. In this arrangement, layers of software are built up from low-level service software, like operating systems and network stacks, to high-level applications (figure 6). Intermediate layers provide domain independent and dependent frameworks that help to contain changes of platform or APIs. As we have seen previously, DSLs can be used to bridge the specification of the application semantic model and the framework semantic model. The use of function blocks inspired in the industrial automation world provides a finer grain of reuse granularity that allows to mix-in fragments of the framework specifications into the application specification.

There are several applications frameworks that can provide common building blocks for different domains (e.g. control systems, distributed services, data processing, hardware interface) that allow developing observatory applications. At the same time provide an isolation mechanism from the underlying layers. Application frameworks can also be seen as a back-end for a DSL. In this case the DSL acts as a veneer on top of the framework components.

Although not necessarily always true, common model building blocks, which are refactored into framework specification blocks, will end up having a common implementation. Ideally it should not be necessary to develop any domain independent framework, as they address common problems to a large family of applications, example of those can be middleware libraries, database management systems, etc.
5. PROCESS

Software requirements often change along the life of a project, especially along the life of a long project. Several development processes under the Agile umbrella have been developed in the last years that try to address this issue. Agile planning is based on features that will be needed in the system instead of tasks. It is easy to plan an entire project using standard tasks without really understanding the system being built. When planning by feature, the team has a better understanding of the system and the plan is easy to be followed by other members of the project. The incremental delivery and refinement of the software features maps naturally the nature of a telescope project, where many of the software systems follow the design of their mechanical counterpart. The same is true about the top-tier software components that must support the operation plans of the observatory, which are also often refined during the life cycle of the project. Moreover the incremental implementation of the features that provide more value or address higher risks helps in the definition of the non-software observatory systems.

Lean principles introduced in Japan manufacturing industry several decades ago keep inspiring the Agile software processes. All the activities related to our software development process are been studied in order to avoid any waste. Repetitive tasks are automated and duplications are eliminated. The goal is to allow every developer to focus in implementing the features that provide more business value to the observatory instead of expending time honoring a document intensive development process.

The division of the system into Subsystems, Packages and Components and the careful grouping of elementary component types help us to manage the complexity, and to understand and visualize both the problem and solution spaces. From a point of view of the organization, components organized in packages represent a common and concrete way to express the scope of the different subsystems. These packages and subsystems will often be developed by different organizations.

As explained in Section 3.1, Features are the basic model element. As part of the specification of a system, they represent the atomic functional requirement. As part of a Function Block collection they define the scope of a component. In an analogous manner the scope of a subsystem is the sum of the scope of its components. Features are thus the atomic planning unit. In an agile process, they are the equivalent to a “user story.” Features are developed incrementally during subsequent iterations.

How to map an agile development process to the “waterfall model” of a general project is always an interesting question. In our planning strategy, the development of every software system is divided in a set of phases, whose deadlines depend on other milestones of the project (e.g. end of detailed design, factory acceptance testing) and that emphasized the nature of the main activities (Figure 7). For every subsystem the features of all the components are assigned to one of the project phases (e.g. development, implementation, acceptance). Business value, degree of risk or milestone compliance criteria are used to assign priorities to features. Subsystems correspond to main Work Breakdown Structure (WBS) elements of the project while Components constitute the elementary elements of the Product Breakdown Structure (PBS).
In addition to components packages, subsystems include management packages (e.g. commissioning plan, documentation) and associated observatory workflow packages (e.g. operation guidelines and processes). As shown in section 2, Features are defined using different DSLs and stored in the semantic model. Modeling tools are used to extract features from the model and to import them into Greenhopper, our agile management tool. Similarly, features are the basic functional requirement and are also imported into the GMT requirement management tool Cockpit [6]. Other project compliant documents will be also generated from the model (e.g. unit test plan, commissioning plan).

Lastly, another DSL has been defined to capture the development process, this way, every aspect of the system is captured in the semantic model including system engineering and management aspects. The development process is defined as a set of Workflows. Some of these workflows include tasks that will trigger the generation of some of the process artifacts (e.g. the complete documentation set of a subsystem).

![Diagram showing the development phases: Design phase, Construction phase, Integration phase, and their associated features.]

**Figure 7 - Features are delivered incrementally across all phases**

### 6. CONCLUSION

The complexity of the GMT system, the distributed nature of the project organization, the required efficiency of the observatory operations, and the need to manage technology evolution while achieving increased levels of productivity, represent challenging factors that have to be addressed with the careful choice of modeling techniques, architectural design, software and hardware platform and development processes.

Tests on our prototype platform will continue during the rest of the year. As technologies are tested they will be incorporated into the baseline of the software and controls preliminary design.

### 7. ACKNOWLEDGEMENT

This work has been supported by the GMTO Corporation, a non-profit organization operated on behalf of an international consortium of universities and institutions: Astronomy Australia Ltd, the Australian National University, the Carnegie Institution for Science, Harvard University, the Korea Astronomy and Space Science Institute, the Smithsonian Institution, The University of Texas at Austin, Texas A&M University, University of Arizona and University of Chicago.

### REFERENCES


