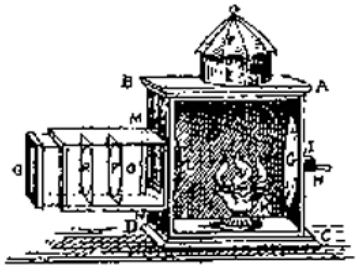

Magic Lantern System Architecture

Version 0.1
February 7, 2007

ML-ARCH-SYS-0.1

WORK IN PROGRESS

Notice

Document Status Sheet

Document Control Number:	ML-ARCH-SYS-0.1			
Documnet Name:	Magic Lantern System Architecture			
Revision Releases:				
Date:	February 7, 2007			
Status Code:	Work in Process	Draft	Issued	Closed

Description of Document Status Codes

Work in Process
An incomplete document, designed to guide discussion and generate feedback, that may include several alternative requirements for consideration.

Draft A document in specification format considered largely complete, but lacking formal review by company/team members. Drafts are susceptible to substantial changes during the review process.

Issued A stable document, which has undergone rigorous company/team review and is suitable for product design, development, and testing.

Closed A static document, reviewed, tested, validated, and closed to further engineering change requests.

Revision History

Revision	Revision Date	Author	Description of Changes
0.1	June 2, 2006	Mark S. Millard	Inital Draft

Table of Contents

1 Overview

Magic Lantern is a interactive, title development platform for authoring highly interactive titles, such as interactive TV applications, games, educational programs, and point-of-information/sales kiosks. It is being created specifically for the real-world needs of professional title development teams as they break new ground in the development of real-time interactive titles.

The Magic Lantern authoring system is being developed with an understanding of how the top professional development teams are currently creating titles. It is designed to streamline development and collaborative processes so that title developers can focus on enhancing content, performance and interactivity. The Magic Lantern development environment will run on Windows and UNIX workstations; for title playback, the Magic Lantern architecture facilitates efficient deployment of a title to one or more platforms, such as an interactive TV set-top box, a Pentium-based PC, or game console.

The Magic Lantern architecture establishes a complete foundation for developing high-performance interactive titles without restricting developers' ability to customize title elements or processes. Its object-oriented design and extensible tools framework encourages the incorporation of specialized add-ons and title templates.

1.1 Components of the Magic Lantern Authoring System

The Magic Lantern Authoring System includes the following components:

- Authoring Tools
- Authoring Framework
- Digital Workprint (DWP)
- Runtime SDKs
- Title Targeting and Optimization Tools

Figure 1-1 on page 4 shows the relationship of these system components.

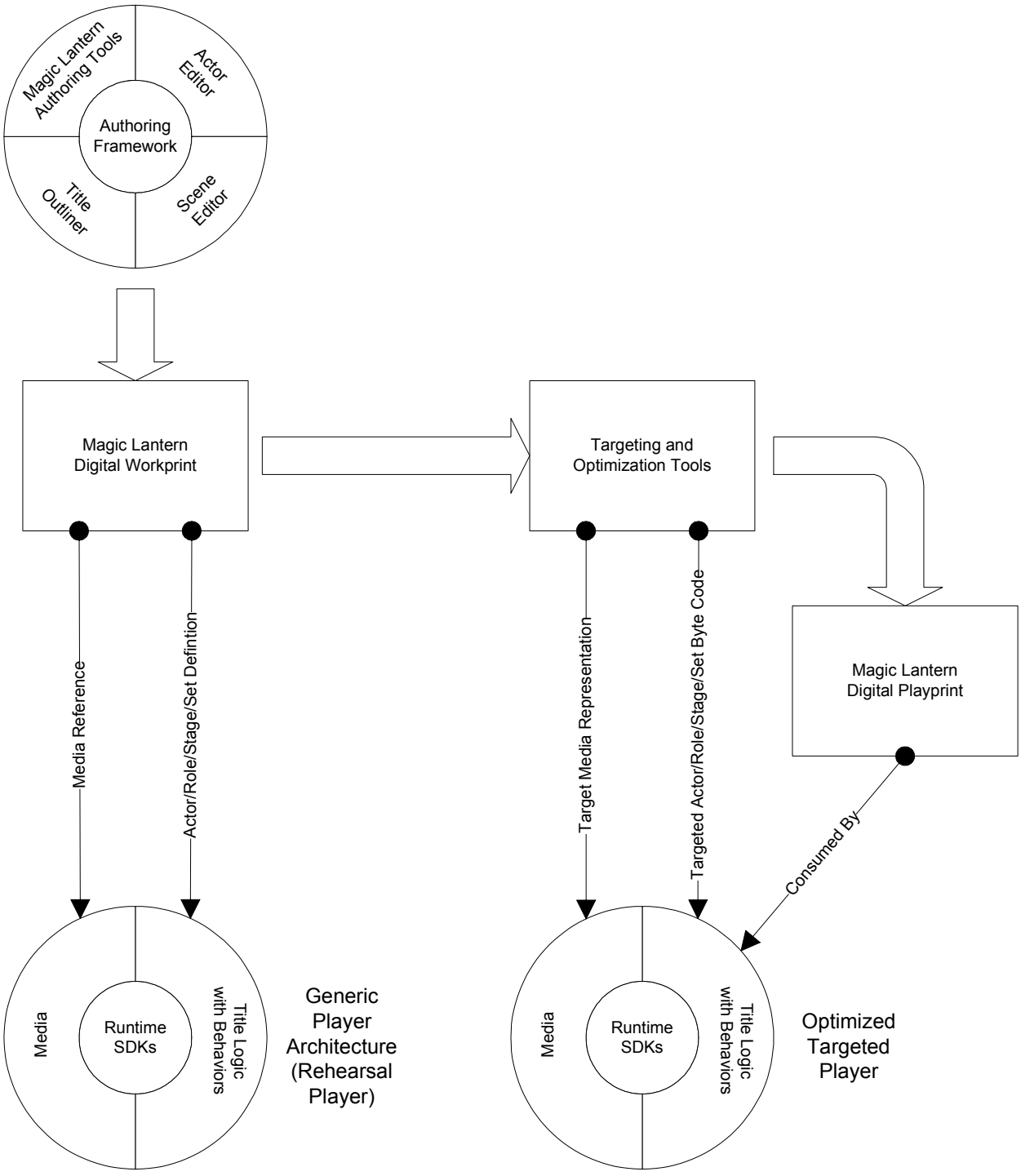

Figure 1-1 *Magic Lantern Authoring System Components*

1.1.1 Magic Lantern Authoring Tools

The Magic Lantern Authoring Tools include the Scene Editor, the Rehearsal Player, the Title Outliner, and the Actor Editor. These tools comprise the title development interface for the entire development team, including programmers, title designers and graphic artists. They allow title artists and designers to view, tune, and change the title and the title elements without an extensive knowledge of the underlying software. Changes that are made with the tools can be saved directly to a Digital Workprint (DWP), the Magic Lantern title specification file format, without compiling.

1.1.2 Magic Lantern Authoring Framework

The Magic Lantern Authoring Framework provides a layered architecture for the Magic Lantern authoring tools and facilitates the creation of additional custom windows and commands for the tools. Third-party tool providers can incorporate specialized tools into the framework without needing a deep understanding of the Magic Lantern software. The Magic Lantern Authoring Framework provides an interface to the DWP, the Rehearsal Player, and the Magic Lantern core libraries.

1.1.3 Magic Lantern Digital Workprint (DWP)

The DWP is the Magic Lantern title specification file format that facilitates title deployment to multiple platforms. The DWP specifies all the resources that accompany a title and is a repository for saved information about the development of a title. The DWP is designed to be an open standard and is based on XML technology that can be extended by title developers and third-party developers.

1.1.4 Magic Lantern Runtime SDKs

Magic Lantern includes C++ and Java classes and runtime libraries for a variety of target platforms. These SDKs provide routines for I/O, audio, graphics rendering, 2D and 3D objects, applying kinematics, and math. In addition, behavioral libraries are included for designing title element behaviors.

1.1.5 Magic Lantern Title Targeting and Optimization Tools

The title targeting tools transform the DWP and associated code resources into its executable version for a specific target platform. They include tools for controlling the targeting process, media translation, color management, and title compilation. The platform representation of the DWP and the title assets is known as the Digital Playprint.

1.2 How Magic Lantern Fits into the Title Development Process

Figure 1-2 on page 6 shows the title development process using the Magic Lantern Authoring System.

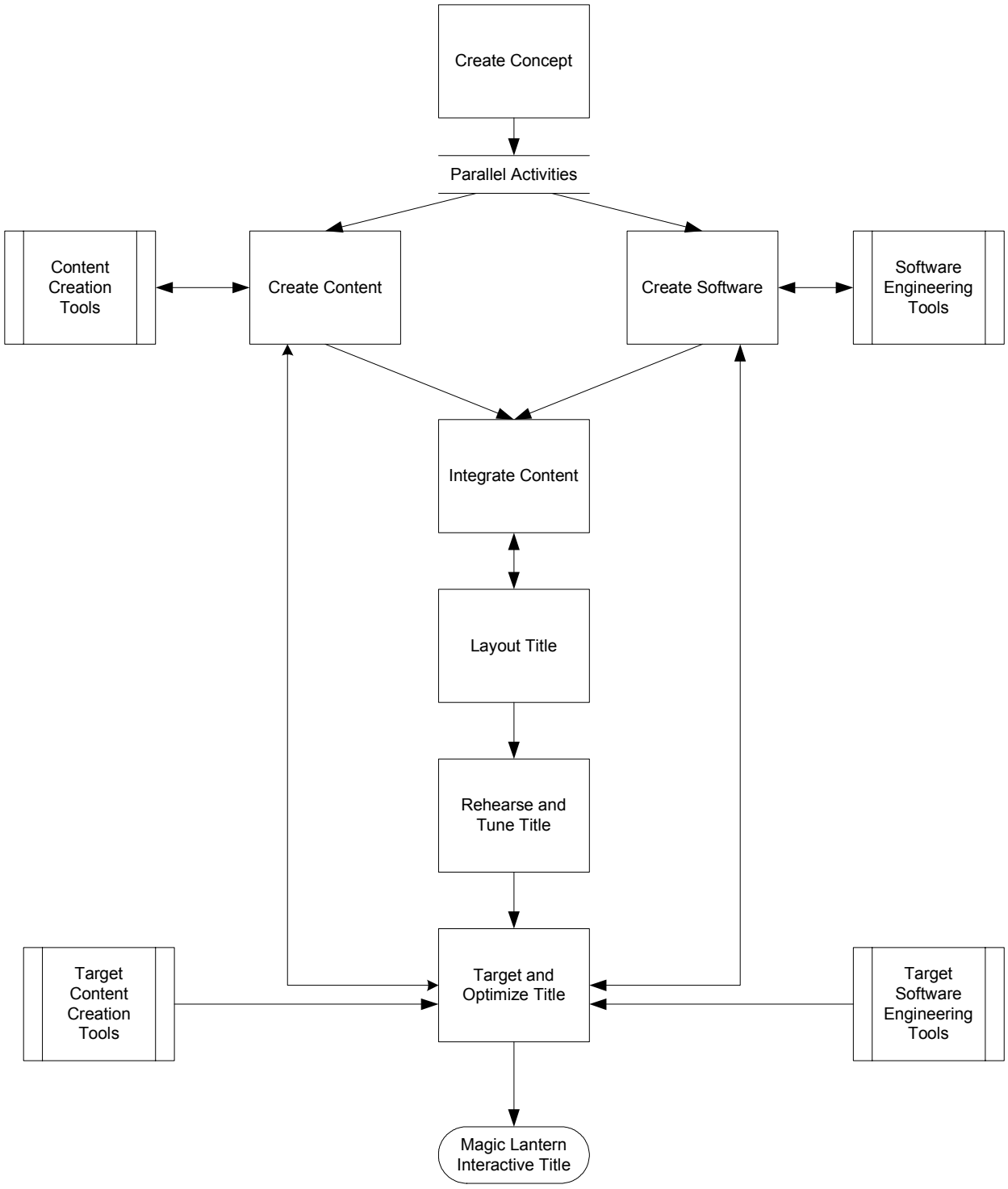

Figure 1-2 *Title Development Process with Magic Lantern*

Magic Lantern offers tools that support the following tasks:

- Content Integration

- Rapid Development and Prototyping
- Title Layout and Tuning
- Debugging
- Targeting and Optimization
- Integrating and Creating "in-house" Tools

1.2.1 Content Integration

Content creators can easily integrate diverse content elements into their title using the Magic Lantern supported data types. Magic Lantern supports files in multimedia formats for 3D models, geometry, animation, 2D images, sound, and video. Magic Lantern offers very tight integration with common multimedia formats.

Supported file formats include:

- Image File Formats: BMP, GIF, JPEG, PNG, RGB
- Audio File Formats: AIFF, WAV, MIDI
- Video File Formats: AVI, Quicktime
- 3D Geometry Formats: Alias | Wavefront, SoftImage, 3DMAX, SGI Inventor and VRML 2.0

1.2.2 Rapid Development and Prototyping

The performance capabilities of current PC and UNIX platforms in conjunction with the Magic Lantern architectural structure lend speed to prototyping and efficient title development. Programmers can build upon the support libraries to prototype title objects and save them to the DWP. This means that Magic Lantern prototypes are not "disposable" efforts that have to be thrown out when title development begins in earnest. Successful prototypes become the basis of the final title and can be continuously improved.

1.2.3 Title Layout and Tuning

The Magic Lantern Authoring Tools make it easier for every member of the title development team to do title layout and tuning. The Scene Editor is used for composing interactive worlds, navigating through scenes, manipulating the objects in the scenes, and creating the DWP. Throughout the development process, titles-in-progress can be quickly viewed in the Rehearsal Player window and edited on the fly to test new ideas. This allows non-programming team members to interactively view and revise artwork, scene layout, and actor properties without assistance from the programming team.

1.2.4 Simulation and Debugging

The Magic Lantern Rehearsal Player can be attached to a debugger so that the title can be run during the debugging process.

Targeted players can be tethered to platform-specific simulators to test and debug a title in context of the constraints implied by that platform. For example, an XBOX player can be tested using an XBOX simulator.

1.2.5 Targeting and Optimization

The Magic Lantern targeting and optimization tools generate the auxiliary runtime data structures that accompany the title code for a specific target. This facilitates iterative testing and optimization of the title on the target platform. Title developers can target the title, test the performance, and then, when necessary, write code to optimize performance for the different targets. The Magic Lantern targeting tools make it more efficient to test the title at the earliest stages with the target audience and iteratively modify the title throughout development.

1.2.6 Integrating and Creating "in-house" Tools

Magic Lantern is being designed from the outset to be an easily extensible system. This allows title developers to use the Authoring Framework API to incorporate their own specialized tools into the system.

2 Glossary

actor
The basic unit of action and presentation within a title.

actor group
See **cast**.

actor property
A C++ or Java member variable that is exposed to the title development team via the authoring tools.

assets
The code and data used to build and run a Magic Lantern title. Assets include the title executable and dynamic software modules as well as the media (that is, 3D models and textures, audio, images, and video).

behavior
Any procedural element that makes an actor move or otherwise change state. Although behaviors can be cyclic and predictable, such as pre-scripted animations, in Magic Lantern the term is typically used to describe more lifelike and complex actions by actors.

behavior package
A modular component for adding behaviors to an actor.

camera
A view into a **3D set**.

camera actor
An actor used to vary the camera view of the **3D set**.

cast
A set of actors that are loaded into memory at the same time.

Digital Playprint (DPP)
A runtime translation of a **Digital Workprint**. The Digital Playprint is consumed by a **Runtime Player**.

Digital Workprint (DWP)
A repository for saved information about the development of a title and which specifies all the resources that accompany a title.

DLL
Acronym for Dynamic Linkable Library

DPP
See **Digial Playprint**.

DSO
Acronym for Dynamic Shared Object.

DWP
See **Digial Workprint**.

media reference
An object used to reference the representation of media across multiple platforms. It is a pointer to an outside media file, such as a graphics image or an audio file.

package
A Magic Lantern component used to add functionality to an actor class.

player
A **Rehearsal Player** or a targeted **Runtime Player**.

pull method
A method of communicating between an actor and its **role** where the **role** explicitly asks the actor for any changes.

push method
A method of communicating between an actor and its **role** where the actor pushes its results to the **role**.

Rehearsal Player
A Magic Lantern program that plays a title on a PC or UNIX workstation. The Rehearsal Player plays the title using dynamic data structures and allows for midstream property changes from the outside, such as from an actor editor. From an interface perspective, the Rehearsal Player has a protocol that other programs can use to start, stop, pause, and change properties. The Rehearsal Player can be activated from the Scene Editor.

Runtime Player
A Magic Lantern program that plays the title using static, mastered data structures. The Runtime Player has no extensions for allowing an outside agent to modify or control anything happening within the title. The Runtime Player loads media groups and actor definitions from a Digital Playprint. A Runtime Player refers to a player that runs on a specific target, such as a PC, game console, or interactive TV set-top box.

role
The software module that handles the actors' platform-dependent presentation on a **stage**.

set
An abstract view of a collection of actors that are presented on a **stage**. A set is used to manage a presentation, such as a 3D scene-graph or 2D windowing environment.

stage
The set of services on the target platform, such as rendering, input, sound, physical modeling, proximity and collision detection, and event handling.

targeting process
The Magic Lantern process that generates an interactive title for its distribution media. The targeting process takes the **Digital Workprint** and its associated media assets as input and creates three output products: the **Digital Playprint**, the translated media, and the executable.

title
The application created with the Magic Lantern authoring tools.

3 Magic Lantern Architecture

3.1 Runtime Architecture

The Magic Lantern Software Development Kit (SDK) consists of four major classes of objects:

- Actors
  Actors specify the platform-independent implementation of the behaviour and interaction of each autonomous element in the title.
- Roles
  Roles specify the implementation of each actor's presentation for a particular platform.
- Sets
  Sets are resource managers that are usually specific to a particular type of media, such as 3D, 2D, audio, etc.
- Stage
  A stage is a machine and operating system (OS) abstraction for sets that manage shared resources and creates the entire frame presentation.

Each object has a specific role in the execution of a title as each gains control of the processor sequentially during each frame of execution. For maximum efficiency, Magic Lantern objects are written in C++ and compiled to native code. Magic Lantern objects may also be written in Java for Java-based targets (e.g. OCAP or DVB-MHP). New functionality is added through subclassing and composition. For simplicity and efficiency, Magic Lantern classes use only single inheritance.

Figure 3-1 on page 11 shows how the main Magic Lantern objects are related to each other.

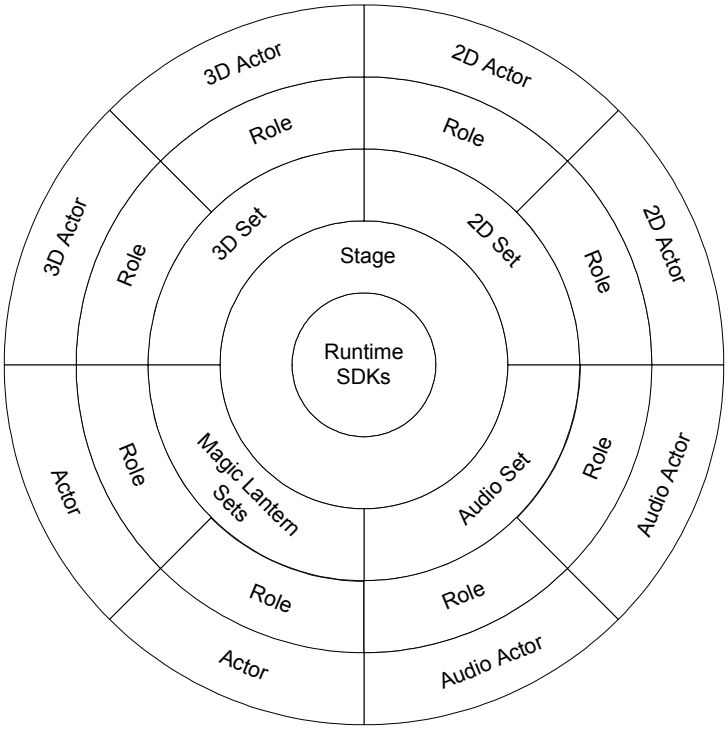

Figure 3-1 *Main Magic Lantern Objects*

3.1.1 Actors

An actor is a simple C++ object that is the basic unit of autonomous action within a title. Actors implement the simulation and behavioral aspects of a title element independent of its presentation. Actor code is intended to be independent of the target platform. This makes it easier to develop a title that is portable to multiple platforms.

Each actor usually has a crorresponding Role object, described below, that is responsible for its presentation in the title. Figure 3-2 on page 12 shows a view of a Magic Lantern Actor with Behaviors in relationship to its Role.

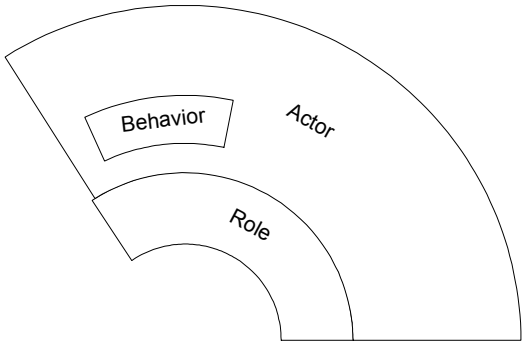

Figure 3-2 *View of Magic Lantern Actor with Behaviors in relationship to its Role*

All the “characters” in a Magic Lantern title are actors. For example, in a driving game, the car, the road, a tree alongside the road, and an on-coming car are separate actors. In addition, sounds can be actors. Thus, the hum of the car accelerating and the explosive sound of a crash could each be generated by a different actor. Cameras for 3D scenes can also be actors that can have behaviors, such as a following behavior. There can be some actors which may have no visible or audible representation in a title. For example, actor objects can function as database managers or collision detection managers.

3.1.1.1 Components of an Actor

An actor consists of the following elements:

- property and non-property member variables
- functions and initialization routines
- an actor definition file (a subset of a Digital Workprint)
- a reference to its role (if any)

A Magic Lantern actor is compiled into a Dynamic Shared Object (DSO) or Java Jar package. This permits actors to be dynamically loaded into an executing title in the authoring environment.

3.1.1.1.1 Properties and Member Variables

The internal state of an actor can be implemented as ordinary C++/Java member variables. Some of these members can be exposed through the authoring tools; these are called "properties" and they are designated in a special way in the class implementation. Actor properties allow other members of the title development team to edit the actor's state in the Magic Lantern tools without additional programming modifications. This can be particularly useful to the team's title designers and artists and can minimize requests for iterative programming revisions. Team members can use properties to define and test different physical characteristics and behavioral characteristics for an actor. For example, a title designer can tune the performance of a vehicle experimenting with various settings of its exposed acceleration property.

Different versions of media can be designated for an actor by establishing "media references" as properties. A media reference is a pointer to an outside media file, such as a graphics image or a sound file. Using media reference properties, artists can use the Magic Lantern tools to independently update actor media or adapt it for specific platforms.

Non-property members can be any data type supported by C++, while property members are constrained to certain Magic Lantern data types. For ease of programming for different platform math requirements, Magic Lantern defines a special data type that can be compiled as either a floating-point value or a fixed-point value.

3.1.1.1.2 Functions

Actor member functions implement the various behaviors of the actor. Because the object implementation is based on C++/Java, the actor programmer can define any number of member functions. Each actor constructor mst be callable from a class static function that expects a common constructor interface. These class static functions are placed in a statically initialized table at runtime. Multple constructors with different interfaces are permitted within Magic Lantern. There must be, however, a constructor that has the no-arguments interface (default constructor).

There are also two initialization routines that are typically implemented by an actor class. The first is a function called *initClass()*. Title programmers write an *initClass()* procedure to register the actor class and properties with the Magic Lantern tools. The tools call *initClass()* which then calls back to registration functions within the tools. The *initClass()* function is compiled out of the actor code for targeted titles.

The second initialization routine, which is optional, is called *init()*. If an actor's init function is present, Magic Lantern will call it after the actor has been initialized from the DWP (during rehearsal) or the Playprint (for targeted titles). This allows the actor to do instance-specific modification of the property values that were set by the DWP or Playprint.

3.1.1.1.3 Actor Definition Files

The actor definition file defines the various aspects of each actor for the tools. The title programmer creates this actor definition file so that the actor can be tested and modified in the tools. The actor definition file is in the DWP format and is typically included in the title DWP files. It defines items such as the actor property names and types, the location of the actor header file and DSO, the actor parent, and so on. These files are not used in a targeted title; they exist only for the sake of the authoring tools.

3.1.1.1.4 Role Reference

Roles are paired with actors to handle the actors' platform-dependent presentation. The role is responsible for displaying the actor. Actors contain a pointer to their role; this pointer can be used to access the role's API which the actor uses to push state changes over to it. As mentioned previously, some actors do not have roles.

3.1.1.1.5 Property Carriers

A property carrier is an object designed for communication between actors and roles. Property carriers are specialized building blocks that carry out actor requests by operating on role internals.

3.1.2 Roles

Roles are the system components responsible for media delivery, such as 3D rendering. As such, they are typically platform-dependent. Any given actor could be paired with one specific role on the authoring workstation (e.g. Linux) yet later be paired with a different role on a targeted platform (e.g Microsoft XBox or Sony Playstation). For example, a Linux-based role might do 3D rendering using an OpenGL library, whereas the PC-based role may use one of the commercially available rendering libraries for the PC platform (e.g. Renderware, DirectX, OpenGL).

Magic Lantern provides roles for 3D rendering and 2D rendering (images and movies). Audio presentation is done by direct communication with the audio sets. Current role classes include:

- MleRole (abstract base class)
- Mle2dRole
- Mle2dImageRole
- Mle3dRole

Roles can also be constructed from scratch by programming in C++ or Java. This approach is appropriate when the media being presented needs to be handled in a special way, such as media-specific level-of-detail control.

As with actors, roles contain conditional compilation structures that accommodate the compilation differences needed for either the rehearsal player or a targeted title. For example, for rehearsal, roles contain additional memeber functions used by the tools to provide hooks for placement, scaling, and orientation. Since the tools do not exist in the targeted title, however, these functions are not required for runtime title playback. Magic Lantern uses standard conditional compilation techniques to achieve this runtime code reduction.

3.1.3 Sets

Sets provide services for their attached actors and roles and are usually specific to a particular type of media: 3D, 2D, audio, etc. In general, they can be thought of as resource managers, where a 3D set manages a shared scene-graph, a 2D set manages layering of items in the frame buffer and an audio set manages synchronizing on the hardware output device. Each set implements a specific API that is used by the roles it manages.

The Set base class implements an initialization function and an *attach()* function. The latter is used to attach roles to sets and its implementation differs for each set type. Some sets need only maintain a collection of roles that use its services, other sets need more structure, such as a hierarchy. Sets may have more than one implementation, using different rendering libraries.

Sets used for graphics include a camera control API. The precise implementation of this API is, of course, dependent o the particular rendering library it uses.

The following collection of set subclasses is provided by Magic Lantern:

- MleSet (abstract base class)
- Mle3dSet
- Mle2dSet
- MleMidiSet
- MleAudioSet

3.1.4 Stages

A stage is all the things that can potentially differ from platform to platform, such as rendering, input, sound, physical modelling, proximity and collision detection, and event handling. A stage refers to the collection of services on the target platform. There are three stages that will initially be accessable through the Magic Lantern tools: the Inventor stage, the DirectX stage and the J2ME stage.

3.2 Runtime Activity

3.2.1 Execution Model Overview

The execution model that contols the sequencing of Magic Lantern titles is a cyclic process repeated once per frame. Each **actor**, **role**, **set**, and **stage** receives control of the processor, in that order. By default, on each frame, all of the **actors** execute before any of the **roles**; all of the **roles** execute before any of the **sets**, etc. This scheduling order is not strict, however, since an **actor** may explicitly pass control to its **role** and that **role** may explicitly pass control to its **set**. Programmers can also reorder or augment the scheduling.

The idea behind tha **actors**-*then*-**roles**-*then*-**sets**-*then*-**stage** ordering is that first the interactivity is computed (**actors**), then the presentation is computed (**roles**), then the various resource controllers gain control (**sets**), then the entire frame presentation is created (**stage**).

Figure 3-3 on page 16 shows the Magic Lantern execution model.

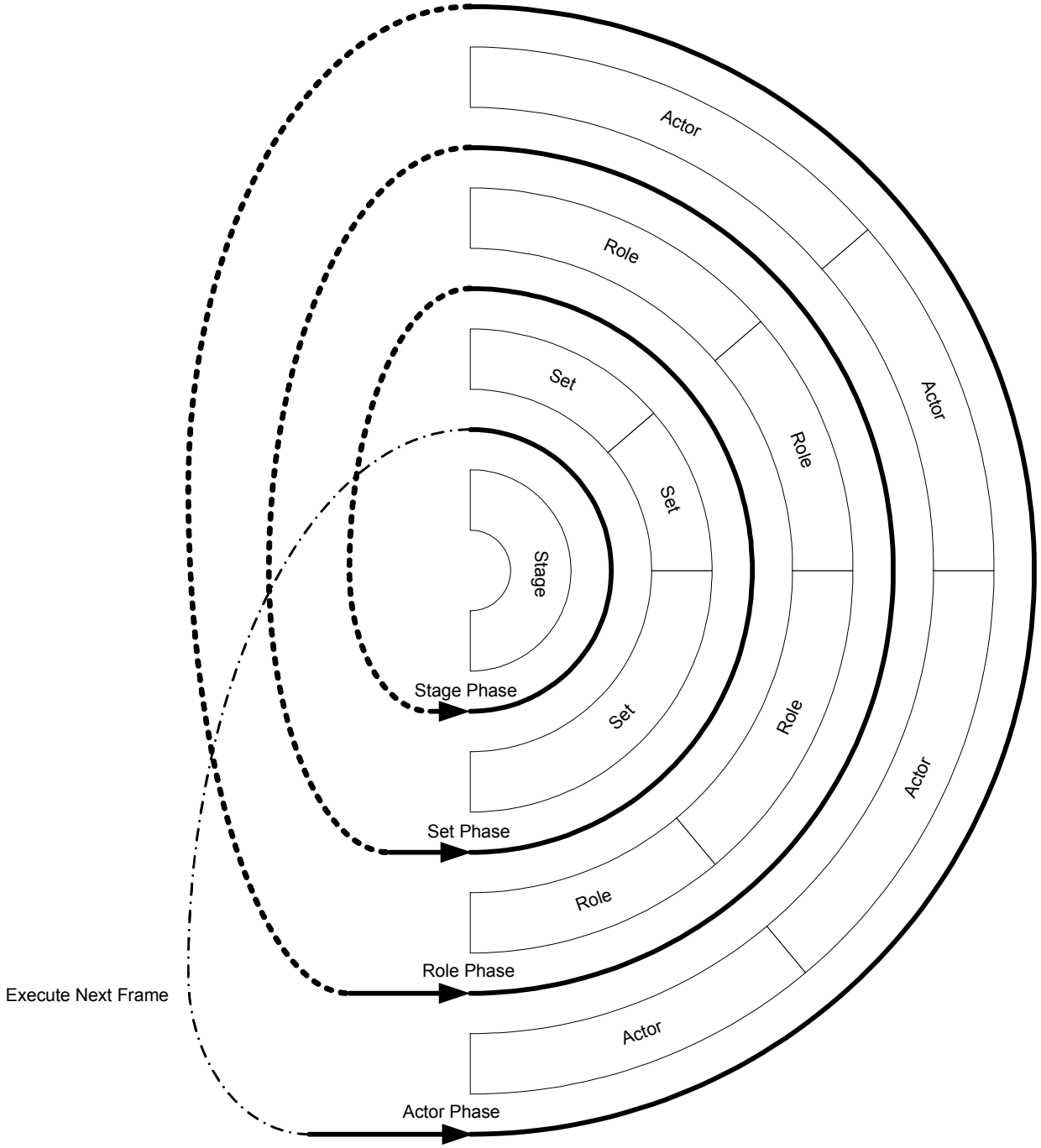

Figure 3-3 *Magic Lantern Execution Model*

The implementation of the main loop of the title that causes this scheduling behavior is dependent on the type of underlying support for scheduling provided by the player platform operating system.

If the target system is completely event driven, then the scheduling of the **actors**, **roles**, and **sets** must be triggered by a timer or user event, and the callback for that event calls the Magic Lantern function scheduler to handle running **actors**, **roles**, **sets**, and **stage**. Precisely how frames are cycled depends on the system, but the effect must be to sequence main elements in the order described above and then repeat that sequence for each frame.

If the target OS or monitor does not "own the main loop" then the title *main()* must.

Pausing a title, as an example, can be implemented by scheduling a 'Pause' **actor** with a special phase of the scheduler and then using a different main loop (or registering a different scheduling callback with the native OS).

3.2.2 Title Start-up

In runtime mode, the *main()* program is very simple and performs the following tasks:

1. Initializes the global title environment structure.
2. Loads the "boot group" - the start-up **actor group** (**cast**).
3. Enters the main loop.

3.2.3 Actor Groups (Cast)

A **group** is a collection of **actors** that is loaded into memory at the same time. As the title progresses and the need for new collections of **actors** arises, new **groups** can be loaded into memory.

Groups may contain **actors** that are associated with any number of **sets**. Upon loading, if a required **set** does not exist, the loader code will create it. This allows entire scenes within a title to be created in a single operation. As an option, the caller to the group loader may supply a registry object that will be filled with references to all of the **actors** that were created when loading the **group**. These references can later be used to delete the entire **group** of loaded **actors**.

Registries have other uses within the Magic Lantern system. A registry can be used for coordinating the behaviors of **actor** collections. For instance, a collection of **actors** that must respond to an external event might register themselves with a particular registry. Then user-supplied code could traverse the registry in response to that event.

3.3 Behavior Packages

A behavior refers to any procedural element that makes an **actor** move or otherwise change state. Behaviors can be cyclic and predictable, such as pre-scripted animations. Typically, however, the term behaviors in Magic Lantern is used to describe lifelike and complex actions by actors such as improvising or reacting to the environment.

Behavior packages are a suggested programming methodology in Magic Lantern that enable title developers to implement **actor** behaviors. Magic Lantern real-time behaviors will initially include a simple vehicle package that can be used for flying or surface-bound objects, lifelike or mechanical **actors**.

Magic Lantern will provide sample **behavior packages** that can be used in conjunction with one another and with custom written packages. For example, a realistic simulation of flocking or herding behavior for a flock of seagulls or a group of marathon runners could be achieved by combining the Separation, Alignment, and Cohesion packages. The combination of these packages would result in a group of **actors** moving in concert, yet each with an individual awareness of its position in relation to other members of the group and the ability to adjust accordingly. Each seagull or runner would be defined as a vehicle with the Vehicle package, with the seagulls' package defining them as flying objects and the runners packages specifying attachment to a surface (such as the ground).

Packages can also be combined to create complex unscripted behaviors for non-biological title elements, such as planes and cars with realistic banking and acceleration properties.

The sample behavioral packages in Magic Lantern focus on the movement of an **actor** around its world, such as basic steering behaviors and collision avoidance, detection and reaction models. Trajectory, not proximity to an object is the most important criterion for obstacle avoidance.

Magic Lantern also will provide simple packages that title developers can invoke from their own action selection layer, where the **actor** makes choices between conflicting goals. For example, Magic Lantern will include a sample *MlePursuer* **actor** which chooses at each frame whether its highest priority should be avoiding obstacles, maintaining separation, or pursuing the nearest **MleWanderer** actor.

3.3.1 Scheduling Actor Behavior

An **actor** is the basic unit of action in a Magic Lantern title, and behaviors are the implementation of that action. Behaviors are member functions that the title programmer writes for each **actor** class and then invokes periodically. Running the behavior functions gives the actor its autonomous behavior. Usually, these behavior functions will run once per frame or once per simulation step, if those rates are not coupled.

Title authors have the option of coding the entire behavior inside one function. Or, the function can implement some of the functionality itself and it can call library functions or other behavioral components to implement the remainder of the functionality. These behavioral components may be other member functions written for the **actor**. They may also be member functions of packages that are attached to the **actor**.

3.4 Digital Workprint (DWP)

The Digital Workprint is a simple object system with containership relations that has both an in-memory API and a XML file format. Items in the DWP are instances of C++ classes derived from a DWP base class (*MleWorkprintItem*). The base class defines the basic capabilities for all DWP items: identity, traversal, structure editing, and input/output. DWP items include **scenes**, **sets**, **groups**, **actors**, and **properties** as well as other resource and attribute specifications.

Title developers and third party developers can create new types of DWP items, which can represent new kinds of resources and attributes, by subclassing existing items. These new types are loaded as Dynamic Shared Objects (DSOs) for recognition by existing applications. The Title Outliner (described on page XXX) provides a view of a title's DWP items.

3.5 Authoring Tool Architecture

3.5.1 Introduction to the Magic Lantern Authoring Framework

The Magic Lantern Authoring Framework is a layered architecture for both the Magic Lantern authoring tools and for building in additional command sets or tools that are tailored to the user's requirements. **Tools**, which are at the topmost layer, provide an organization and user interface for the commands. **Commands**, at the next layer, do most of the work in the application. The final layer in the authoring tools is composed of **Managers**, which provide general but necessary services, such as selection, cut, copy and paste.

Figure 3-4 on page 19 shows the relationship between Tools, Commands and Managers. Figure 3-4 on page 19 also shows the interaction of the Authoring Framework with the Digital Workprint.

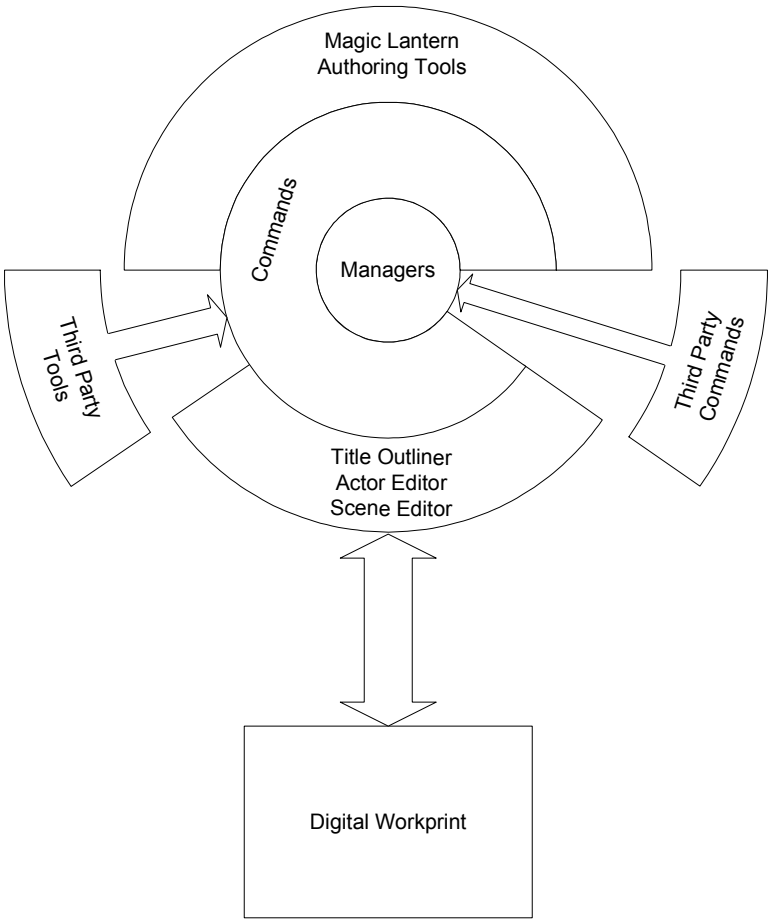

Figure 3-4 *Magic Lantern Authoring Tools and Authoring Framework*

The Authoring Framework is built on top of the Eclipse SWT interface library. This library provides the framework for developing interface components. Programmers have access to all the functionality to which they are accustomed when writing Eclipse applications.

3.5.1.1 Tools Layer

Tools define window user interfaces and possibly new commands. Tools are invoked from commands. The Magic Lantern tools include

- the Scene Editor,
- the Rehearsal Player,
- the Actor Editor, and
- the Title Outliner.

Commands from third party applications can be plugged into the Magic Lantern tools. Third party tools can be integrated into the Magic Lantern Authoring Framework.

3.5.1.2 Command Layer

Commands do most of the work in the application. All actions that affect the player or the Digital Workprint (DWP) must be done by a command. Commands are lightweight extensions to an application that implement a user function. They are invoked from a menu item, button or another command. Commands may or may not be undoable. Commands can have a user interface; they will show up as a menu item or icon. Finally, commands are scriptable using the Magic Lantern Scripting interface.

Title programmers can add new functionality to the framework via a tool Dynamic Shared Object (DSO). Functionality that does not require a complete window user interface can be added as a command DSO.

3.5.1.3 Manager Layer

Managers in the Magic Lantern Authoring Framework provide general services, such as selection, cut, add actor, and save Digital Workprint. Each manager provides a set of convenience functions for managing DWP items or controlling the Rehearsal Player. Data can be stored and shared between tools via the managers.

3.5.2 Introduction to the Magic Lantern Authoring Tools

The Magic Lantern Authoring Tools provide the user interface for the Magic Lantern authoring system. Their purpose is to expose meaningful title authoring tasks to non-programming members of the title development team, and to maximize the expressive capabilities of title designers and artists through high-level operations and usability. The authoring tools are also commonly used by programmers for title rehearsal and debugging. As previously described, the main Magic Lantern Authoring Tools are the Scene Editor, the Rehearsal Player window, the Title Outliner, and the Actor Editor.

3.5.2.1 Scene Editor

The Scene Editor is the primary editor for building, editing, and examining a Magic Lantern title. The Scene Editor allows any member of the title development team to edit and view the title. The Scene Editor presents the initial state of a scene in the title as it exists in the Digital Workprint.

Magic Lantern ready actors, which include title components and other media, can simply be dragged and dropped into a Magic Lantern scene in the Scene Editor. Both programmers and non-programmers can use the Scene Editor to populate a scene and position objects in the title. The scene-in-progress can be viewed instantly through the Rehearsal Player window. Any changes made in the Scene Editor can be saved to the DWP without recompiling, which allows non-programmers to make meaningful changes to the title-in-progress without technical assistance. New DWPs, scenes, sets, and cast (a collection of workprint itmes that can be loaded at runtime as a single unit) can be created in the Scene Editor.

Figure 3-5 on page 21 shows an example of the Scene Editor.

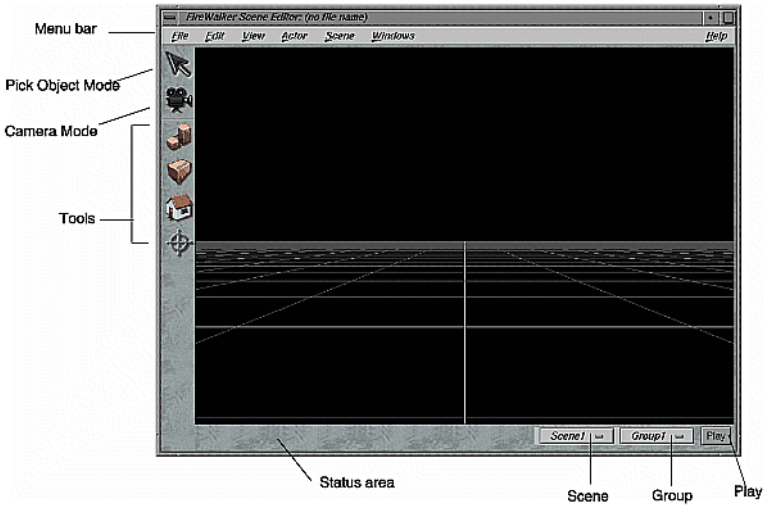

Figure 3-5 *Magic Lantern Scene Editor*

Title team members can rehearse while authoring by clicking the Play button to open the Rehearsal Player window (described on page XXX).

3.5.2.2 Rehearsal Player

The Rehearsal Player window allows title designers and programmers to view the title in real-time. It provides title statistics and is integrated with the Actor Editor (described on page XXX).

Figure 3-6 on page 21 shows an example of the Rehearsal Player window.

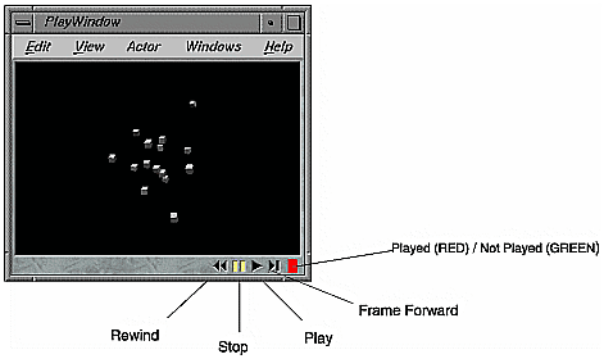

Figure 3-6 *Magic Lantern Rehearsal Player*

The Rehearsal Player window is the user interface to the Magic Lantern Rehearsal Player. When the title is executing in the Rehearsal Player, its actor's properties can be edited via the Magic Lantern Actor Editors. Changes made in the Rehearsal Player are not saved to the DWP; they must be explicitly saved through the Scene Editor. This is analogous to making changes in a source code debugger (not permanent) versus making changes in s source code editor (permanent).

The Rehearsal Player has a protocol that the authoring tools can use to start, stop, pause, and change properties. Internally, the Rehearsal Player can be integrated with a debugger (such as Microsoft's Visual C++ tools) for debugging titles.

Figure 3-7 on page 23 shows the static view of a Magic Lantern title during rehearsall.

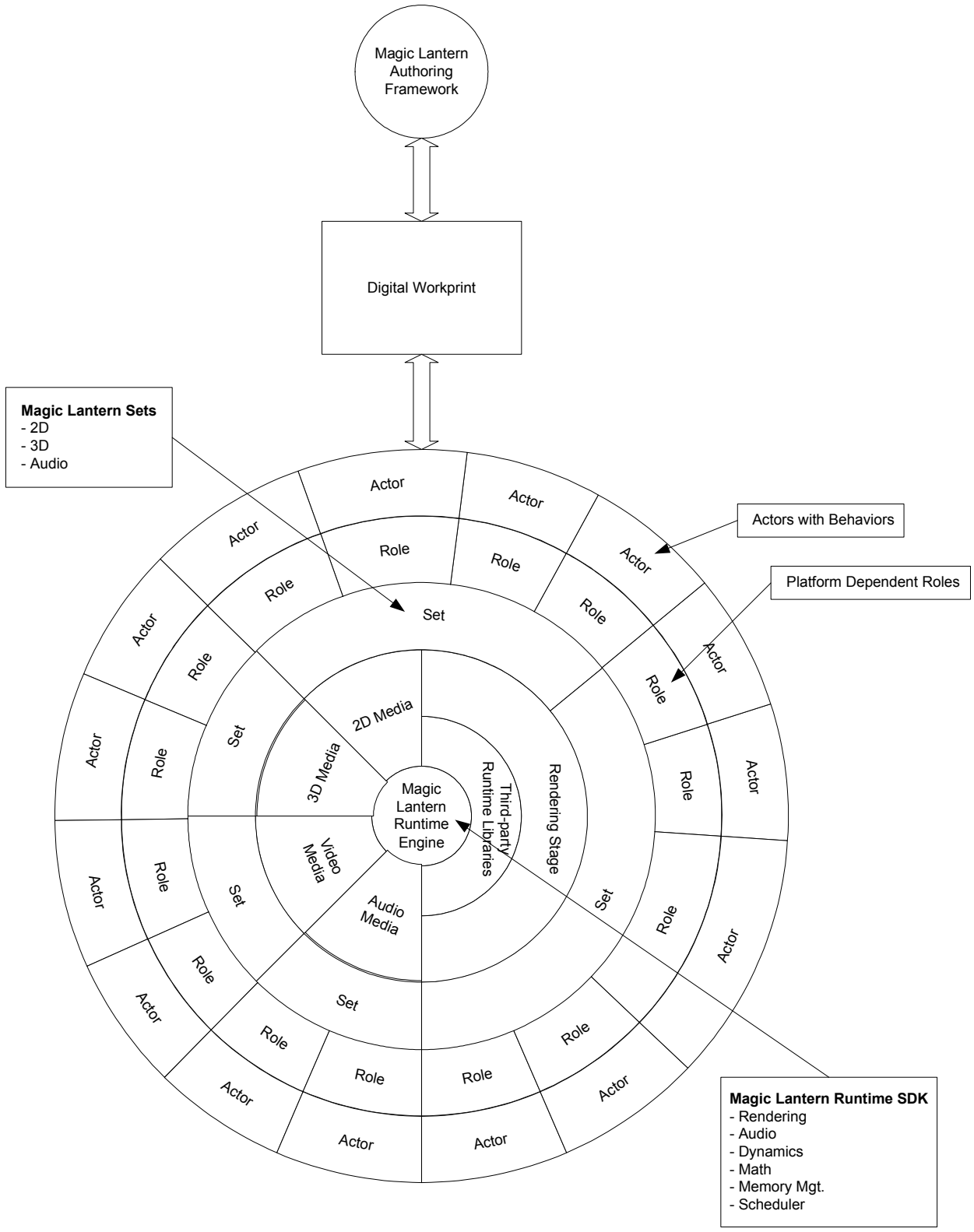

Figure 3-7 *Static view of a Magic Lantern title during rehearsal*

3.5.2.3 Title Outliner

The Title Outliner provides an inventory view of the items in the DWP. The Title Outliner is used to browse the DWP inventory and find items in a title based on various search criteria.

The Title Outliner provides a hierarchical view of the DWP. It opens the specified DWP and displays its top-level components, such as scenes, sets, and casts, which in turn can be selected for viewing of subsequent levels. At the lowest level, double-clicking on a component opens the corresponding Actor Editor.

Figure 3-8 on page 24 shows an example of the Title Outliner.

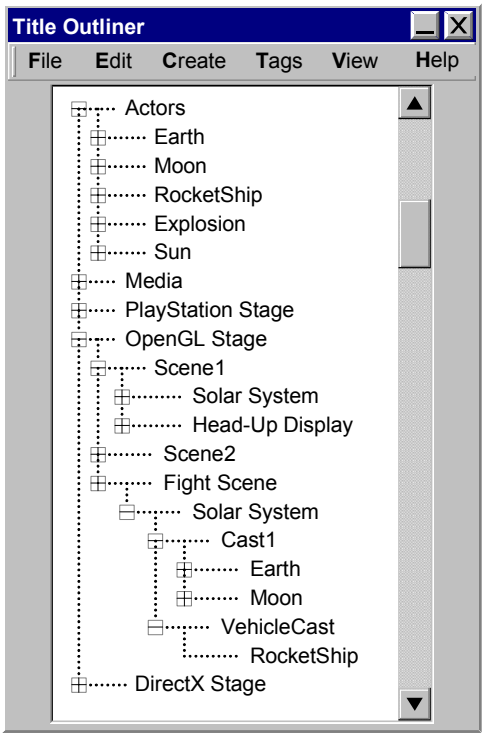

Figure 3-8 *Example of Magic Lantern Title Outliner*

The Title Outliner can be used to search for items by name or type. It is also useful for locating and editing invisible or hard to find actors.

3.5.2.4 Actor Editor

The Actor Editor displays the selected actor's editable property list as defined by the title programmer. (See Properties and Member Variables on page XXX).

When creating the editable property list for an actor, programmers can use both the default properties based on the actor's type (for example, a sound bite actor has properties different from a graphics actor) and any other actor variables that they wish to expose.

Any member of the title development team can access the Actor Editor for a particular actor by double-clicking on that actor in either the Scene Editor, the Rehearsal Player window, or the Title Outliner. Property values are set and modified using th Property Editors called from the Actor Editor. Changes made to property values via the Scene Editor will be saved to the DWP; changes made to property values via the Rehearsal Player can be viewed, but not saved.

Figure 3-9 on page 25 shows an example of an Actor Editor.

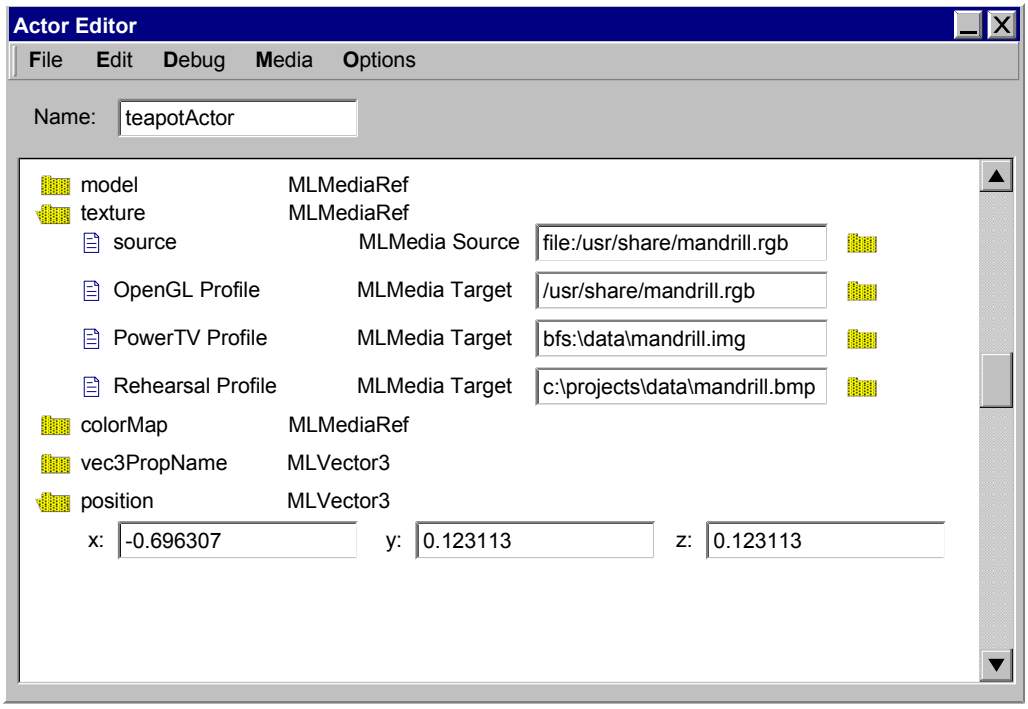

Figure 3-9 *Example of Magic Lantern Actor Editor*

In addition to the generic actor editor provided by Magic Lantern, title developers can easily write a customized actor editor for an actor with XXX or can plug in third-party actor editors through the Magic Lantern Authoring Tools Framework.

3.6 Targeting

The Magic Lantern targeting process is part of the development of an interactive title. This process uses the title components and makes them playable on a specific target, such as a PC running Windows DirectX, a game console or a set-top box.

The targeting process can be used during all phases of title development. At the start of title development, it is used to prototype the title on the target, to explore the development limits, and to view the media on the delivery platform. As development progresses, the targeting process is necessary to help refine the title, keeping it efficient. When the development is complete, targeting is the final step in producing a platform-specific title that is ready to be transferred to the distribution media.

3.6.1 Overview of the Targeting Process

The targeting process uses several programs that help prepare an interactive title for its execution on its target device. The targeting process takes the Digital Workprint (DWP) and its associated media assets as input and creates three output products. These three principal products of the targeting process are:

- the Playprint

  The Digital Playprint contains compiled casts (groups of actors) and references to translated media. It is the platform representation of the DWP and the assets associated with an interactive title.

- the translated media

  The translated media is the associated title media assets translated to a form appropriate for the target platform.

- the title executable

  The title executable is the executable program containing all of the title logic.

3.6.2 Understanding the Targeting Process Steps

There are several key steps in the targeting process. Title developers can also bypass the default solution and implement their own custom scheme for each step.

These are the key targeting process steps:

- The title developer converts media to produce properly formatted media files.
- The title developer generates a target Makefile for building a specific target Digital Playprint (a binary file format that is accessed by the Magic Lantern Runtime Engine).
- Magic Lantern generates chunk files, which are restructured DWP information, to create the Digital Playprint. There are four types of chunk files: cast chunk files, media reference chunk files, scene chunk files and set chunk files.
- Magic Lantern generates runtime look-up tables that contain pointers to constructors for title elements.
- Magic Lantern produces the Digital Playprint file. As an intermediary step, a Digital Playprint layout script may be generated to drive the build process. Using the layout script, the Playprint generation tool copies the chunk files into a single binary file, the Digital Playprint.
- The title developer creates the executable title by compiling the authoring code and the generated targeting code and then linking it against the Magic Lantern Runtime libraries.
- The title developer generates the distribution media (such as a CD) with all title components placed in a format native ot the target. This is the final step of the targeting process. This setp is performed outside of Magic Lantern.

Figure 3-10 on page 27 illustrates the major components of the targeting process. It shows the steps outlined above and the flow of different key pieces of data.

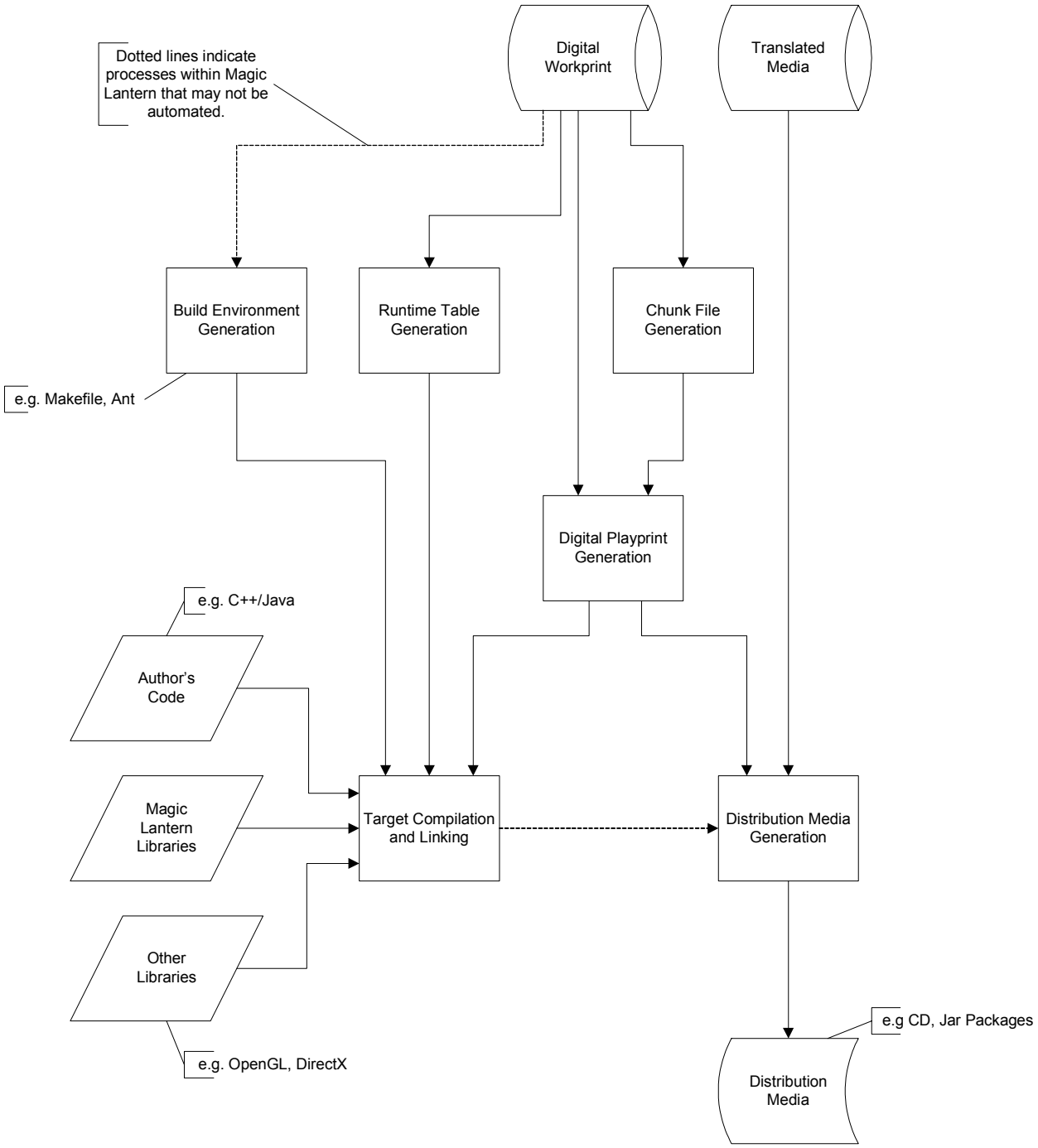

Figure 3-10 *Magic Lantern Targeting Process*

3.6.2.1 Digital Playprint Generation

The Digital Playprint Script Interpreter assembles the Digital Playprint from chunks stored in individual files. These chunks are the compiled **actor group** chunk files (a **cast**), **media reference** chunk files, **scene** chunk files and **set** chunk files. A Digital Playprint Generation Script drives the Digital Playprint Script Interpreter. The language for the script is TCL-based. It uses commands to lay out the actual Playprint. The Digital Playprint Script Interpreter also generates primitives for converting symbolic Rehearsal Player names into numeric runtime player names. It places these runtime player names in files that must be compiled and linked against the Runtime Engine libraries.

The Digital Playprint Generation Script is created with the Digital Playprint Script Generator tool that reads the Digital Workprint. The script is customized or replaceable by the title programmer.