RIN: A Declarative Specification for Interactive Narratives over Rich Media

Naren Datha
Microsoft Research India
narend@microsoft.com

Joseph Joy
Microsoft Research India
josephj@microsoft.com

Eric Stollnitz
Microsoft Research Redmond
ericsto@microsoft.com

ABSTRACT
Rich media technologies like streaming video, gigapixel panoramas, and terapixel maps are becoming broadly available on the Internet. Although rich media offer wonderful opportunities for creating experiences that have strong interactive and narrative elements, the ways in which users experience these media are widely disparate, involving a plethora of similar-yet-different web sites and mobile applications, each with its own proprietary rendering logic, data formats, and back-end services.

In this paper we introduce RIN —a declarative specification for orchestrating interactive, cinematic narratives that thread through an extensible set of rich media experiences. RIN's XML (and JSON equivalent) markup makes it straightforward to represent complex, cinematic fly-throughs that fluidly compose diverse media technologies, including gigapixel panoramas, terapixel online maps, traditional paginated documents, and data visualizations, choreographed together with audio, text, and video. RIN introduces a concise representation of the logical state of an experience, called *Small State*, and the specification of the evolution of this state over time, called an *Experience Stream*. These abstractions along with others introduced in this paper enable a concise, declarative specification for a wide range of rich media experiences. We believe RIN can serve as the foundation for the next generation of standards-based rich media on the Internet, enabling a variety of new and compelling scenarios.

1. INTRODUCTION
Rich media experiences on the web have moved far beyond text, images, and video. There are numerous immersive web sites built using technologies like Flash, Silverlight, HTML, and Java [1], [11]; as well as applications available from proprietary app stores run by technology companies Apple, Google, Microsoft, and others [7], [13]. These experiences are developed with technologies that use a combination of custom client-side rendering and server-side storage and computation—examples include gigapixel panoramas [10], Deep Zoom images [17], streaming video, online maps, and data visualizations such as PivotViewer [18]. Integrating any of these experiences into a web site or application with today's technologies requires custom programming that relies on a menagerie of organically grown libraries and web services, and in the case of applications targeted at application stores, the proprietary APIs of the targeted devices.

Besides the barrier to building rich interactive experiences being high, most such experiences on the web today exist in what we call "islands of interactivity"—you can navigate to a particular site, or download a particular application, and interact with each experience in isolation.

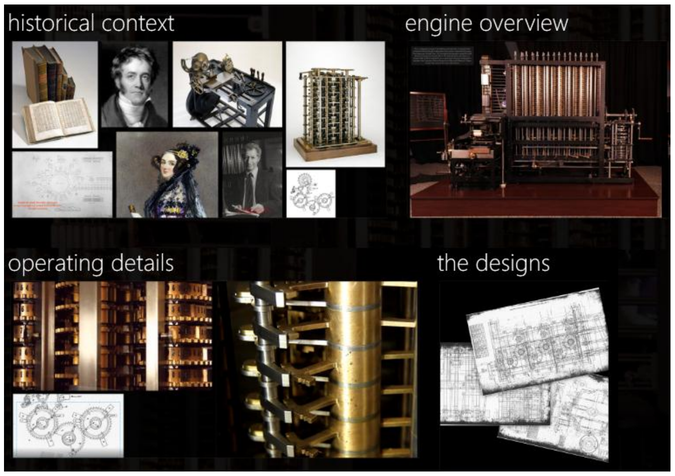
We feel strongly that an enormous amount of information is latent in the way people put rich interactive content through its paces, add their own insights, and tell their own *stories*, employing ancillary media to augment these stories. We are by no means the first to make this observation. World Wide Telescope pioneered the concept of interactive tours that guide you through its particular realm, that of the solar system and the universe [9], [11]. Google Earth [8] also supports tours over the Earth, and Chronozoom [1] supports tours over the time span of the universe. The architecture of each of these applications is done in a domain-specific manner, and they are neither composable nor extensible.

In this paper we present RIN, which stands for Rich Interactive Narratives. RIN has the following goals:

- *Unify the worlds of cinematic narrative and interactive exploration.* Enable rich, cinematic-style nonlinear narratives, threading through an open-ended set of interactive experiences that include the latest breed of rich media technologies.
- *Take a document-centric approach, as opposed to a programming-centric approach.* The hybrid narrative-interactive experiences should be defined declaratively, with transparent and concise logical structure that reflects the semantics of the content.

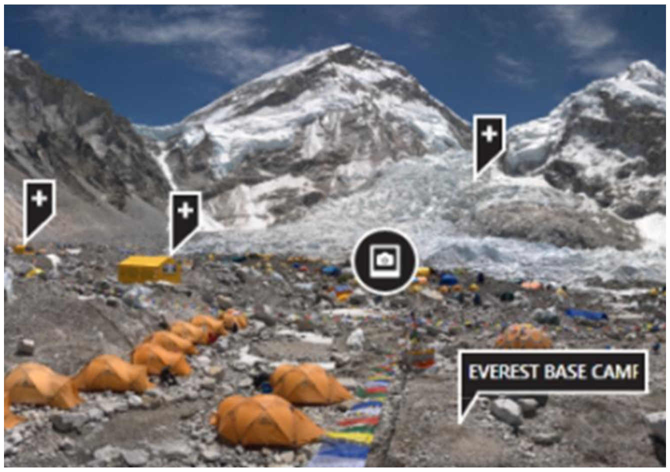
To the best of our knowledge, RIN enables for the first time:

- *A concise definition of the logical state of a wide range of rich media experiences.* This concept can be used to "jump" to a particular state within an experience. When used as a deep reference into rich content, RIN's notion of state serves as a generalization of the HTML "<a>" anchor tag to rich media.
- *A declarative way to script a path through an open-ended set of interactive experiences, forming an interactive narrative.* The interactive narrative enables both a cinematic "play" experience (generalized to support multiple timelines) as well as the ability to pause and explore the underlying interactive experiences.
- *A platform-agnostic architecture for rendering these experiences.* Both proprietary and standards-based platforms are supported.



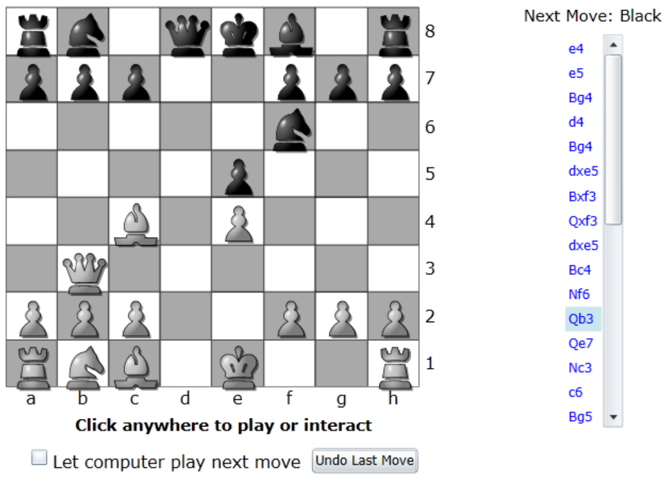
(a) The Babbage Difference Engine No. 2 experience is composed of multiple narrative threads that weave through a media collection providing historical context, a gigapixel image of the machine, videos of the engine in operation, and annotated scans of the original design documents. Verbal narration provides context while the underlying visual media is scripted to change fluidly, highlighting the topic at hand. For example, when describing the engine's operation, the narrative pans and zooms through the gigapixel image, then transitions seamlessly into a video showing a part of the engine in operation. The user may pause and explore at any time.

Image credits: Computer History Museum, Mountain View, and *Science Museum, London.*



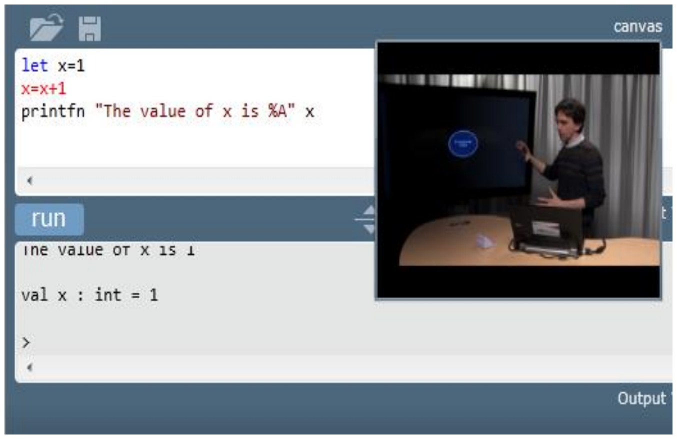
(b) The Everest experience is a world of linked gigapixel panoramas, with embedded "artifacts" that trigger various stories composed of other media. For example, if the user taps the second "+" artifact from the left, a sub-narrative is launched that zooms into the boxy orange tent, and transitions to a 360-degree panorama of the *inside* of the tent. The tent happens to be a high-altitude gallery of images, and tapping any of these images launches a sub-narrative that emerges into another panoramic landscape, accompanied by a cinematically compelling narrative and musical score.

Image credits: David Breashears and Glacierworks


Next Move: Black
8
7
6
5
4
3
2
1
a
b
c
d
e
f
g
h
Click anywhere to play or interact
Let computer play next move
Undo Last Move

```
e4
e5
Bg4
d4
Bg4
dxe5
Bxf3
Qxf3
dxe5
Bc4
Nf6
Qb3
Qe7
Nc3
c6
Bg5
```

(c) This is an annotated account of a famous chess match, complete with audio commentary. In contrast to just watching a video, the user can select from multiple endings, and can interrupt the narration at any time to move the chess pieces. The underlying experience includes a built-in chess engine that can simulate the moves of an opponent. At any point, the user can resume the narrative, in which case the pieces all return to their positions when the narrative was paused and the story continues.



```
let x=1
x=x+1
printfn "The value of x is %A" x
```

```
The value of x is 1

val x : int = 1

>
```

(d) This experience is a programming lesson, structured as a two-person verbal dialog between an instructor and a student. The dialog threads through a fully interactive interpretive programming language session. As the dialog progresses, the underlying interactive session evolves, i.e., code snippets are inserted and run as part of the narrative. The user can interrupt and try her own code variations at any time. In fact, at certain points, the user *has* to step in and participate by entering or modifying code and pressing "run." Additional media (such as the inset video pictured here) is brought in to illustrate certain points in the lesson.

Image credits: Microsoft Developer Network

Figure 1: A sampling of experiences built using RIN

We have built tools to author RIN content, and a Web-plugin-based player that renders RIN content. Work on an HTML5 player is under way. The tools have been used to create a wide variety of content, a sampling of which is presented in Figure 1. The examples in the figure cover widely different domains, ranging from a world consisting of linked gigapixel panoramas to an interpretative programming language session. Despite this diversity, they all share a set of core abstractions, they are all expressed in the same XML-based RIN markup language, and, in fact, they were all authored using the *same* GUI authoring tool (Figure 2). A single player is capable of interpreting and rendering all of these examples, dynamically loading specialized components as required.

RIN XML content (and JSON equivalent) is defined at a fundamentally higher semantic level than presentation markup languages like HTML and CSS, and is platform-agnostic. The content is structured as a document that contains the abstractions summarized below:

Experience	contains the information required to define and populate a particular rich media experience (such as a particular instance of a panorama with embedded content). This information typically includes URLs pointing to data sources as well a string that uniquely identifies the pluggable components required to render the experience.
Small State	represents the logical state of an experience at a point in time. This core abstraction enables the declarative manipulation of rich media experiences.
Experience Stream	represents the scripted evolution (i.e., animation) of a rich media experience as a series of time-stamped snapshots of Small State.
Screenplay	contains compositional and timing information that specifies how multiple Experience Streams are orchestrated into a single linear narrative sequence. A RIN document may contain multiple screenplay instances that thread through the same set of underlying experiences. While each of these individual screenplays defines a *linear* narrative sequence, they may be linked together into an arbitrary topology, or triggered by a user interacting with embedded content, producing a user experience that contains both linear and nonlinear aspects.
Segment	is a self-contained unit of RIN content, analogous to a "page" in the HTML setting. Contains all the information required to render a set of rich media experiences as well as to play linear narrative sequences that thread through them. A Segment contains one or more Experiences, Experience Streams and Screenplays.

This paper introduces the core RIN data abstractions and the execution model for rendering RIN content.

2. MOTIVATION AND GOALS

While declarative formats exist for text, images, and graphics primitives, most so-called "rich media" experiences are incorporated into end-user applications and web sites by imperative programming to APIs exposed by libraries. Examples include: online maps such as Bing Maps and Google Maps; gigapixel panoramas [10]; and data visualization controls such as Pivot Viewer [18] and D3 [2]. In order to build an experience that combines such experiences in nontrivial ways (beyond simple mash-ups) programmers write custom code that ties events to proprietary API calls. This process is pervasive and alternatives are hardly considered. This is not surprising given that there is no viable alternative to integrating these rich media technologies, which often involve complex client-side code interacting with server-side components. For example, smoothly panning and zooming through a panorama requires client-side rendering logic that dynamically pulls in and transforms pre-computed image "tiles" from a server. Furthermore, these rich media technologies have evolved organically, each with its own API set.

Unfortunately, this situation results in rich media applications and web sites that are opaque in nature—either locked up in a tar ball of interdependent client- and server-side scripting, or packaged as one-off applications. One disadvantage is the incredible difficulty of combining rich media into an application or web site, with significant custom development overhead and a result that is often tied to a particular technology or platform. Another significant disadvantage is that the resulting applications become black boxes, with little opportunity for federated use and repurposing of content.

RIN has two chief goals:

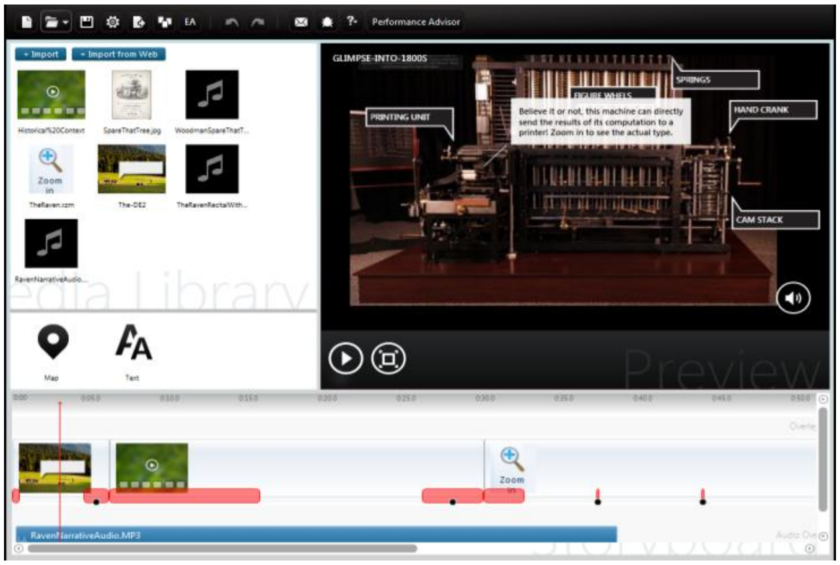


Figure 2: Prototype Authoring Tool. The tool emits XML RIN Segment files. RIN content players are oblivious to the existence of this tool; it represents one way to create RIN content. The tool assembles linear sections of an interactive narrative over rich media. Rich media assets are imported into the area in the upper left, and are assembled into a screenplay using the timeline below. Individual Experience Streams are created by selecting an item on the timeline, manipulating the underlying rich media and capturing snapshots of Small State.

1. *Define a composite form of media experience that supports multiple narratives threaded through a variety of rich media experiences.* The scope of RIN is not arbitrary multimedia experiences, but rather composite experiences that merge cinematic narrative with interactive exploration through an open-ended set of rich media. The user can enjoy a video-like "play" experience, or pause at any time to explore the underlying media and choose alternative paths in a nonlinear narrative.

2. *Take a document-centric approach, as opposed to a programming-centric approach.* There should be a clear separation of concerns between the declaratively specified content and any code required to render the experience. The document should have a transparent and concise logical structure that reflects the semantics of the content, rather than low-level presentation details mixed with scripts or code that calls into a menagerie of libraries.

We believe a document-centric approach with a clear separation of concerns between content and implementation opens up numerous and exciting opportunities for construction, transformation and consumption of the composite experiences, analogous to the diversity of ways HTML markup is used to build web experiences.

3. CORE ABSTRACTIONS

3.1 Experience and Small State

RIN enables an open ended set of rich media experiences to be specified and manipulated declaratively, i.e., *as data*. RIN provides consistent data abstractions that work across diverse experiences. For example, in RIN, paths through an interactive map, a Deep Zoom image and a 360-degree panorama are represented by data that have consistent structure across these different kinds of rich media. Key to enabling data-driven manipulation of rich media experiences is the RIN concept of Small State:

Small State is serializable data that represents partial, logical state of an interactive experience.

Figure 4 illustrates the relationship between an interactive experience and its Small State. The interactive experience (Bing maps in this case) may have considerable *internal* state, and may interact with external services (e.g., the Bing maps search engine and tile server) in order to render the experience. The experience typically interacts directly with the user (e.g., to permit panning and zooming of the map). Small State is a *partial* and *logical* representation of the state of the experience at a point in time. It is *partial* because it does not represent the internal state of the experience in its *entirety*, but rather only that portion of the state that can be manipulated by a user or scripted from within RIN content. A large amount of detail required to bring up the experience is left out of Small State, and is represented by w in the

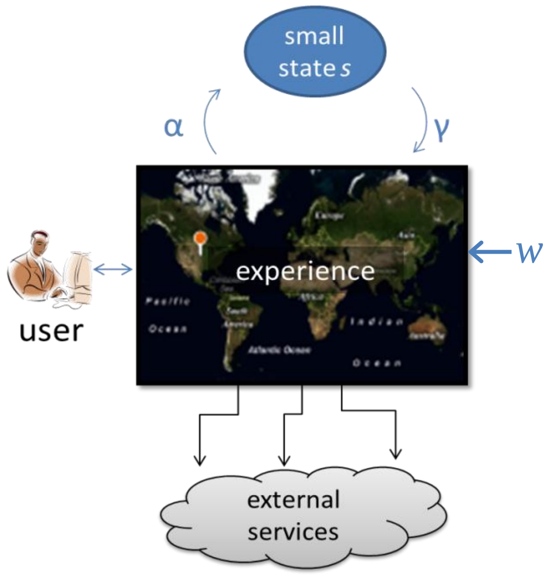


Figure 4: Illustrating the conceptual relationship between an experience and its Small State

figure (w stands for *world* data). This includes all aspects of theming and styling that are not modifiable by the user or narrative script. This crucial separation of concerns enables the concise and elegant manipulation of experiences. Small State is *logical* because it is a semantic representation that captures the high-level meaning for the experience to be in a particular state. For example, RIN defines a Small State quantity known as a *viewport region*, consisting of four real numbers. This piece of data represents a rectangular region, in world coordinates, of an underlying 2D experience that occupies the entire client viewport. In the map example, the numbers represent the geographic boundaries of the visible portion of the map. However, because this single structure defines the world-to-viewport mapping logically, it may be applied to *any* experience that has two independent coordinates, including online maps, panoramas, and 2D images of any scale, as illustrated in Figure 3. In a later section we show how one can compose scripted paths through these media simply by specifying a sequence of viewport regions. Small State may be defined for any experience, including the navigation state of a paginated document, the query state for faceted search over a large collection of items, the state of chess pieces on a game board (Figure 1 (c)) or the evolving code of a software program (Figure 1 (d)).

The relationship between an experience and its Small State is represented symbolically by two functions, called α and γ (see Figure 4). We refer to α as the "state extraction function," since its role is to query the internal state of the experience and extract the corresponding Small State, s. For example, in the case of a map

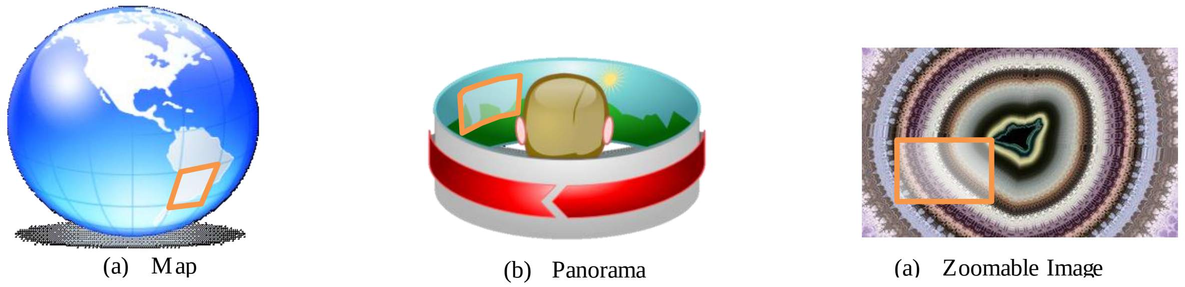


Figure 3: Examples of a *region* mapping, part of Small State, applied to different kinds of rich media. The orange regions define the portion of the underlying media that map to the client viewport.

experience, s would contain the previously-mentioned region mapping. We call γ the "execution function." Supplied Small State s, the γ function swings into action, manipulating the experience to conform to s. In mathematical terms, $\alpha(\gamma(s)) = s$.

Strictly speaking, γ is parameterized by w, the world data, which provides the additional information required to fully render the experience. The execution function γ employs information implicit within the internal run-time state of the experience in order to perform its role. For example, if s contains a viewport region, and the experience is a gigapixel panorama, applying γ causes the underlying panorama to pan and zoom appropriately from its current state until the viewport contains precisely the region of the panorama specified by s. Small State may be used to succinctly characterize the user-modifiable and/or scriptable state of a very wide and open-ended set of experiences. Section 5.1 presents the structure of Small State.

3.2 Experience Stream

Small State represents a *snapshot* of the partial state of an experience. The second core RIN abstraction, Experience Stream, represents the *evolution* of an experience as a function of time:

An **Experience Stream** $s(t)$ is Small State as a function of time.

An Experience Stream is a stream of Small State data. If the execution function γ is applied to this stream, it produces a *scripted* experience, in other words, it scripts or animates an interactive experience along a predefined trajectory or path. Conversely, the evolution of an experience while the user is interacting with it may be represented logically and declaratively as the Experience Stream $\alpha(t)$.

While an Experience Stream is a *continuous* function of time, in practice (i.e., in the RIN specification) it is typically represented by discrete snapshots of Small State, called *logical keyframes*, each with an associated timestamp. The Experience Provider (see Figure 10) is responsible for smoothly interpolating the scripted experience. In other words, the implementation turns a sequence of discrete logical keyframes into a continuous function $\gamma(s(t))$.

Figure 5 depicts three discrete logical keyframes along a timeline, and corresponding screenshots of a fluid flythrough of an image that pans and zooms along a trajectory defined by Small State contained within the keyframes.

Figure 6 shows an abstract representation of Experience Streams. The figure shows two Experience Streams within the same experience, represented as trajectories, i.e., paths, through the state space that represent all possible values of Small State that may be assumed by the experience.

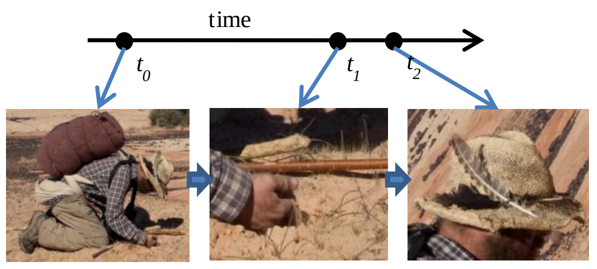


Figure 5: Illustrating an Experience Stream

The state space is typically a high dimensional space whose dimensions could contain real and discrete values. For example,

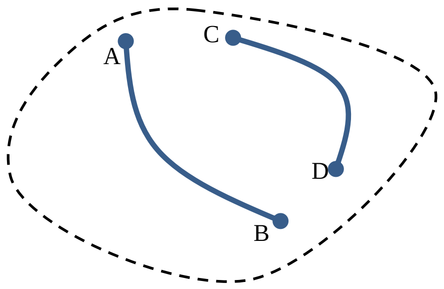


Figure 6: Two Experience Streams within the same Experience represented as trajectories A→B and C→D within the state space of Small State

the camera parameters that capture the world-to-viewport mapping of panoramas, images and maps constitute a 4-dimensional space – one for each of the 4 numbers that represent the region illustrated in Figure 3. Added to these would be additional dimensions to represent various viewing options (for maps, these would include "aerial", "road", etc.)

The relationship between an experience and its Small State has direct bearing on the organization of the software built to author and render RIN content. Experience Provider libraries (see Figure 10) implement the equivalent of α and γ functions, and interpolate smooth paths between the discrete keyframes. Section 5.2 introduces the structure of an Experience Stream.

3.3 Screenplay

RIN supports composition of experiences, and the primary unit of composition is the Screenplay:

A **Screenplay** is an orchestration of multiple Experience Streams along a linear timeline.

Figure 7 illustrates the logical structure of a Screenplay. The boxes ES1-4 each represent an Experience Stream, and therefore correspond to a trajectory in some experience.

The spatial layout of experiences may be explicitly specified (e.g., z-order) in the Screenplay. However most spatial layout details are implicit in the theming and styling of the experience (e.g., a player "Skin" experience that surfaces user interaction controls), and thus form part of world data w.

Figure 8 represents the logical state of the system at the pause point in Figure 7.

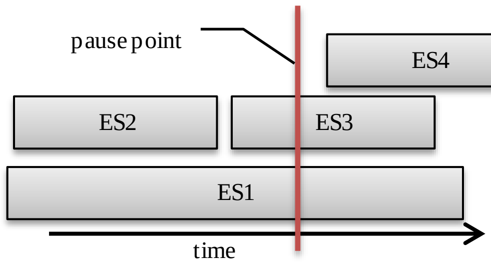


Figure 7: Illustrating Screenplay Structure. ES1-4 are Experience Streams sequenced along a timeline. The user can pause a playing screenplay at any time, represented by the "pause point" in the figure. The user may then directly interact with the experiences, and subsequently resume the narrative.

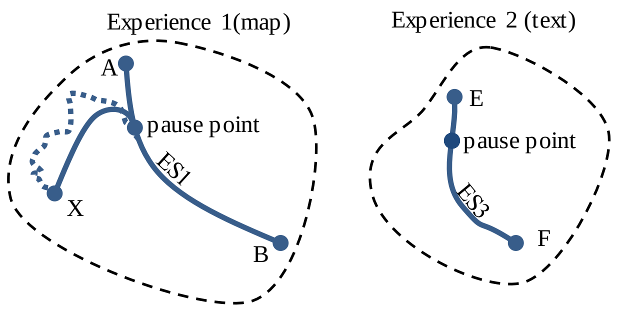


Figure 8: Logical state of the system at a pause point

At the pause point, two Experience Streams are active, ES1 (in experience 1, say a map experience), and ES3 (in experience 2, say a block of text overlaying the map). The dotted line from the pause point to "X" represents the user's exploration path within the map. The user has left the trajectory at the pause point and manipulated the experience to end up at the point where the Small State is represented by "X". (In this example, let us assume that the user's exploration does not alter the state of experience 2.) On resuming the Screenplay, an *on-the-fly* trajectory is generated (solid line from "X" to the pause point) to "swing back" to the correct point in the trajectory in order to resume playing ES1. In the case of experience 2, since its state was not changed, resuming would start exactly at the pause point.

The RIN XML and JSON formats for Screenplays has a simple structure that provides metadata for Experience Streams that occur on the timeline, including time offset and z-order. The RIN Screenplay format is in fact not a core part of the RIN specification. Instead, the reference player architecture (Figure 10) supports pluggable Screenplay Interpreters, and leaves it to these interpreters to read and interpret the Screenplay data. A chief motivation for this architecture is to encourage experimentation in Screenplay functionality, including future exploration of the use of the SMIL 3.0 [4] timing and synchronization format.

Screenplay facilitates composition in the following ways:

- Multiple Experience Streams may be laid out on a timeline, sequentially or overlapping, analogous to media items laid out on a video sequence. Note that each Experience Stream is a scripted path through what is often a fully interactive and rich experience in its own right.
- Screenplays can be launched from within other Screenplays. This recursive encapsulation enables more complex composite experiences to be composed from simpler structures, as well as enables the same assets to be referenced from multiple locations.
- Objects embedded within experiences can trigger launching of Screenplays. This is the primary mechanism to build non-linear narrative topologies.

The above functionality enables a diversity of composite experiences such as those introduced in Figure 1. For example, in Figure 1(b), if the user invokes embedded artifacts such as the '+' object, it triggers the playing of a Screenplay that launches a mini-story. This story starts in the present experience (the gigapixel panorama), brings in additional media components to augment the cinematic feel, and involves a scripted pan and zoom through this experience that in many cases ends in a different panorama altogether. At any point the user can pause and explore, and may launch a different sub-story by interacting with some other

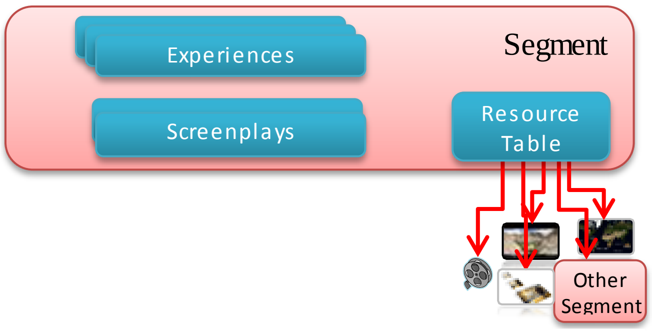


Figure 9: Structure of a RIN Segment

artifact. The structure of our current version of Screenplay is presented in 5.3.

3.4 Segment

Encapsulating all the structures discussed thus far is the Segment:

A **Segment** is the containing unit for experiences, Experience Streams, Screenplays and the resources they reference.

The Segment, illustrated in Figure 9, is analogous to an HTML page in that it is a self-contained unit referencing everything required to present a composite RIN experience. Included in a Segment is a table of resources that provides a level of indirection to all external resources referenced within the Experience Streams. Segment structure is presented in Section 5.4.

4. EXECUTION MODEL

The RIN concepts of Experience, Small State, Experience Streams, Screenplays and Segments are data abstractions for representing interactive narratives over rich media. Bringing these experiences to life requires an execution model that interprets and renders RIN content. The semantics of this model are as important as the declarative data specification. The semantics include important pragmatic considerations such as interpolation with fluid transitions, theming and localization.

RIN content can be created in any number of ways. It may be hand-crafted using an authoring tool, such as the prototype authoring tool shown in Figure 2 that was used to author all content shown in Figure 1. Alternatively, content may be

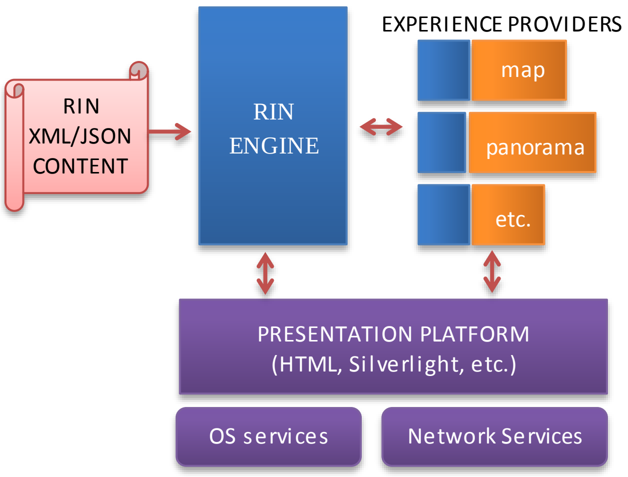


Figure 10: Reference Player Architecture

generated automatically by a program or service, or by dynamically combining hand-crafted and automatically generated content. Regardless of origin, RIN content may be "played" or "rendered" by any implementation of the RIN Engine and Experience Providers that conform to the reference architecture.

Figure 10 illustrates the reference architecture for rendering content. RIN content (in XML or JSON form) is interpreted by the core RIN engine, which dynamically loads Experience Provider libraries as necessary to present the rich media experiences called for by the document. The RIN engine and provider libraries (collectively known as "RIN player") are implemented using appropriate presentation-layer technologies, such as Silverlight, HTML/JavaScript, and potentially iOS and Android APIs.

Instances of the RIN player may be hosted in a diversity of ways, ranging from being "div" elements within existing HTML content to being a stand-alone application.

The blocks colored blue in Figure 10 represent RIN-specific code, while the orange blocks represent existing rich-media controls for experiences such as panorama viewing and online maps. The API to the Experience Provider libraries is the most significant extensibility interface in RIN since the entire user experience is composed of one or more potentially-interacting Experience Providers.

4.1 Player Functions

An instance of a RIN player is bound to a single Segment at any point of time[1]. Once bound, the player has two primary states:

Playing: the content is in scripted mode – it is on "autopilot" playing out a linear sequence as defined in a Screenplay

Paused: the content is paused for user exploration – it is in "manual" mode where the user may explore the content

The following two Engine methods represent the essence of a RIN player:

`play` *screenplayId timeOffset*

`pause` *screenplayId timeOffset*

Either method may be called when the player is in a Playing or Paused state, on the same or different Screenplay. These two functions encompass a *significant* amount of implicit behavior that is not apparent at first glance. A large part of the behavior involves producing fluid transitions when moving from the current state to the desired state.

The Engine and the Experience Providers share responsibility for executing `play` and `pause` with fluid transitions. In fact, the sole reason for both `play` and `pause` to take screenplay ID and time offset as parameters (as opposed to, say, introducing a separate "`seek`" command[2]) is to give Experience Providers the flexibility to craft on-the-fly trajectories customized for the transition. We take a deeper look into transition semantics in Section 4.3. The remainder of this section describes the roles of the Engine and the Experience Providers when processing Screenplays.

4.1.1 *Initialization*

Processing a Screenplay involves the following initialization steps:

1. The RIN Engine identifies and loads the appropriate Screenplay Interpreter—a subcomponent (not shown in the figure) that understands how to interpret the particular format of the Screenplay.
2. The Screenplay Interpreter uses information in the Screenplay and referenced Experiences to identify the appropriate Experience Providers—libraries that know how to render particular Experiences. At the discretion of the implementation, the orchestrator may immediately load all the experience providers required for the screenplay, or may defer loading until just before they will actually be needed to render content.

4.1.2 *Preparing to Transition*

Executing `play` or `pause` often involves switching context to a different Screenplay or a different logical point in the current Screenplay. This entails the following preparatory steps that need to be executed before performing the actual transition:

1. The Screenplay Interpreter analyzes the Screenplay information to determine which set of Experience Streams need to be instantiated given the particular seek point. At its discretion, for performance reasons, the interpreter may look ahead to determine future resource requirements. For example, if switching context to the pause point illustrated in Figure 7, it would determine that Experience Streams ES1 and ES3 are required and may include ES4 because ES4 is required shortly "downstream" from the current point.
2. The Screenplay Interpreter provides the Orchestrator (a subcomponent of the Engine) a list of the Experience Streams that are active at the desired point in the Screenplay timeline.
3. The Orchestrator loads any required Experience Providers, and if it has not already done so, requests that the Experience Providers load the required experiences.

4.1.3 *Play*

The Play operation is called to commence playback of a linear section of a narrative at the specified time offset from the start of the specified Screenplay. Play involves the following steps:

1. The Engine orchestrates the preliminary steps outlined in Section 4.1.2, at which point all needed experiences are loaded with the required Experience Streams.
2. The Engine instructs each Experience Provider to commence rendering a scripted path, as specified by the loaded Experience Stream, starting at the specified time offset relative to the start of the Experience Stream.
3. Experience Providers in turn render their content using an animated path in a manner appropriate for the specific experience. Typically they have already set up interpolation functions using data from the Experience Stream, and when asked to play, they simply render the instantaneous view at each cycle of a rendering loop by computing interpolated values and passing those values to appropriate media-specific API calls. For truly fluid transitions, the Experience Providers may need to create *on-the-fly* transition trajectories that are spliced into existing Experience Streams (such as illustrated by the solid line from "X" to the pause point in

[1] Complex content may be organized as *multiple* instances of RIN Engines each bound to its own Segment, to enable scenarios such as a RIN Segment playing as a piece of content within another RIN Segment. The state of each instance is distinct.

[2] Earlier versions of the RIN reference architecture had a "`seek`" method at the level of the Engine and Experience Providers. This method has since been discontinued in order to enable fluid transitions.

Figure 8). Design guidelines to construct on-the-fly trajectories to deal fluidly with the various transitions are an important aspect of constructing good Experience Providers. Section 4.3 has more details.

4. While `play` is in progress, the Orchestrator may, at its discretion, look ahead in the screenplay to determine future resource requirements, and may preload experiences. (In practice, this is an important implementation detail. Our player implementations aggressively preload media in order to mitigate mid-stream pauses while content is being loaded. This is a generalization of the kind of preloading that happens in audio and video players.)

4.1.4 Pause

The Pause operation is invoked to temporarily halt the progress of a playing narrative, and optionally transition to a different logical time in the same or a different Screenplay. Pause involves the following steps:

1. The Engine orchestrates the preliminary steps outlined in Section 4.1.2, at which point all needed experiences are loaded with the required Experience Streams for the new location (this is a no-op if there is no change in Screenplay and time offset).
2. The Engine instructs all loaded Experience Providers to transition the experiences from their present state (which could be Play or Paused) to a Paused state at the specified Experience Stream and time offset.
3. The Experience Providers in turn disengages any running scripted animations and switches to interactive mode. As with executing `play`, the Providers may need to craft on-the-fly transition animations to fluidly transition the state from its present state to the Paused state.

4.1.5 *Process URL*

A key feature of RIN is that the *state* of a composite experience can be succinctly represented as data. This includes the state of the system in the case that the user happened to have paused and *explored* the content. This data and can be encapsulated into a URL, enabling expressive "deep references" into RIN content. The URL includes the Spreenplay ID, time offset, and optional user exploration state. Section 5.5 contains more details about deep-referencing RIN content.

When presented with such a URL, the player needs to "rehydrate" the state, and this involves the following steps:

1. Load and initialize the segment (Section 4.1.1);
2. Jump to the specified location by effectively executing "**`pause`** *screenplayId timeOffset*"; and
3. Extract any user-exploration Small State from the URL and push it down to the corresponding Experience Providers.

4.2 Experience Provider Interface

The extensibility of RIN comes from the flexibility of the Small State data definition (Section 5.1), as well as the interface between the RIN Engine and the Experience Providers. Two core functions of the Experience Provider interface are analogous to the Engine methods introduced earlier:

`play` *experienceStreamId timeOffset*

`pause` *experienceStreamId timeOffset*

ExperienceStreamId identifies the Experience Stream. *TimeOffset* identifies the logical time relative to the start of the Experience Stream.

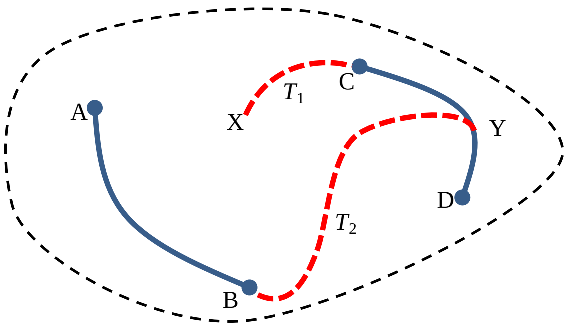


Figure 11: Illustrating fluid transitions T1 and T2

Additional interfaces facilitate inter-experience communication as well as ease the implementation of Experience Providers. For example, the α Small State extraction function (introduced in Section 3.1) takes the form of a callback method where a provider notifies the Engine of changes in the experience state in the form of a Small State update; other experiences can register to receive these notifications. Another significant subset of functionality is the support for theming and localization via dynamic resource resolution. These and additional aspects of the provider interface are documented in the RIN SDK.

4.3 Fluid Transitions

Cinematically fluid transitions are an important part of the overall RIN experience. Experience Providers play a critical role in this, and need to pay special attention to transitions between the playing and paused states and when switching among Experience Streams. Figure 11 illustrates two characteristic kinds of transitions, labeled T_1 and T_2.

Transition T_1 is initiated when the user has paused the narrative and is at explore point X. Some event (typically the user clicking on an object) has triggered this transition, which eventually causes Experience Stream C-D to play. Note that simply *resuming* a narrative from explore point X is a special case of this kind of transition.

Transition T_2 is initiated without user interaction at the end of Experience Stream A-B. This transition is typically launched by the Screenplay Interpreter in response to "end action" metadata present in the Screenplay.

Transitions can be considered Experience Streams that are generated *on-the-fly* by the Experience Provider, taking into account the start and end conditions. End conditions include both the instantaneous experience state as well as *dynamics*, i.e. rate of change. For example, due to dynamics considerations, the transition T_2 depends on whether the narrative is to be in the paused or playing state at the end of the transition. If the end state is Paused then the transition would have an easing function that brings all movement gradually to a stop at the end of the transition. If the end state is Playing then movement would be adjusted to match the rate of change of Experience Stream C-D at the point of attachment.

While fluid transitions provide the best experience, ultimately it is up to implementers of Experience Providers to decide how much to invest in fluid transitions. The simplest approach is to abruptly "jump" to the desired state with no attempt to craft a fluid transition or match end-state dynamics.

5. DECLARATIVE SPECIFICATION

This section presents an overview of the syntactic structure of the core RIN data abstractions. The full details are present in the RIN document specification, the latest draft of which is available on request.

5.1 Small State Structure

The set of possible rich media experience RIN supports is open ended—any experience that can be wrapped in an Experience Provider library that implements the α and γ functions may be incorporated into RIN content. In order to avoid a proliferation of gratuitous custom Small State definitions, RIN factors Small State into a set of sub-structures we call **Slivers**. The RIN document specification includes a foundational set of Slivers. Our goal is that the logical state of a wide range of experiences is represented and manipulated by means of relatively few, well documented Slivers. A second, equally important benefit of Slivers is to enable the specificaion of *partial* updates to Small State. Partial updates may be used for efficient notification of real time state updates, as well as to efficiently represent Experience Streams, as explained in Section 5.2.1.

Slivers are incorporated into Small State in the form of collections of properties grouped by a named *target*. Each target identifies a logical endpoint within a user experience to which the properties are addressed. Small State has the following form:

[XML]

```
<state>
  <target1  (scalar properties) >
    (compound properties)
  </target1>
  <target2  (scalar properties) >
    (compound properties)
  </target2>
  (additional targets and their contained properties)
</state>
```

[JSON]

```
"state": {
  target1: {
    (properties)
  },
  target2: {
    (properties)
  },
  (additional targets and their contained properties)
}
```

In the notation above, `target1` and `target2` are placeholders for specific named targets such as `viewport` (which is the target for all Small State that defines the visual transformation from Experience-specific world coordinates to viewport coordinates).

Sliver properties may be either scalars or compound objects, and accordingly they are present as either attributes or elements in the XML version or as scalars or compound objects in the JSON version.

The following sample of Small State contains a single target (`viewport`) which contains a single compound property (`region`):

[XML]

```
<state>
  <viewport>
    <region>
      <center x="0.2" y="0.3"/>
      <span x="0.7" y="0.5"/>
    </region>
  </viewport>
</state>
```

[JSON]

```
"state": {
  "viewport": {
    "region": {
      "center": {
        "x": 0.2,
        "y": 0.3
      },
      "span": {
        "x": 0.7,
        "y": 0.5
      }
    }
  }
}
```

The `region` property specifies a mapping from the 2D coordinate system of a particular experience (say, a map of the world) to the viewport on the display device. Its `<center>` element defines a point in world coordinates (latitude and longitude), and its `<span>` element defines the width and height of the rectangular region, also in world coordinates.

Note that the same Sliver structures can apply to different kinds of experiences, representing media that are traditionally manipulated using completely different sets of APIs. For example, the `region` property may be applied to any media whose primary world coordinate system is based on a 2D manifold. Figure 3 shows how maps, panoramas, and zoomable images may all use `region` to express the world-to-viewport mapping.

The following example introduces additional targets with a mix of scalar properties:

[XML]

```
<state>
  <appearance displayMode="minimized"/>
  <animation offset="2.0" duration="10.0"
             play="true"/>
</state>
```

[JSON]

```
"state": {
  "appearance": {
    "displayMode": "minimized"
  },
  "animation": {
    "offset": 2.0,
    "duration": 10.0,
    "play": true
  }
}
```

The `appearance` target specifies the style of appearance, such as whether the object should appear in a minimized form.

The `animation` target provides the state of execution of linearly animated media, such as video, audio, or even a RIN Screenplay.

Table 1 lists the current set of standard targets.

Table 1: Standard Sliver Target Definitions

Target	Description
`appearance`	General appearance aspects
`viewport`	World to viewport mapping
`filter`	Content filters (search)
`animation`	State of linearly animated media
`document`	Discrete part of document being displayed
`collectionReferences`	A meta target that associates Small State with named sub-components (see text for details)

`Appearance`, `viewport` and `animation` were introduced earlier.

`Filter` contains properties that define a subset of content that is to be viewed. `Filter` has two properties currently defined, `simpleSearch` and `facetedSearch`. `SimpleSearch` supports a single text query, while `facetedSearch` supports faceted search. A sample `filter` Sliver is shown below:

[XML]

```
<state>
  <filter>
    <simpleSearch query="monuments"/>
  </filter>
</state>
```

[JSON]

```
{
  "state": {
    "filter": {
      "simpleSearch": {
        "query": "monuments"
      }
    }
  }
}
```

`Document` contains discrete information that identifies the specific portion of content being viewed, such as the page number when viewing a multi-page document.

A sample `document` Sliver is presented below:

[XML]

```
<state>
  <document pageNumber="20"/>
</state>
```

[JSON]

```
{
  "state": {
    "document": {
      "pageNumber": 20
    }
  }
}
```

`CollectionReferences` is an advanced mechanism to specify the Small State for *subcomponents* within an experience. `CollectionReferences` is a "meta target" in the sense that other targets may be *nested* within its specification. The special element `itemReference` is used to address a particular item within a collection.

The following example defines the state of objects within two collections, one named "`highlights`" and the other named "`source-1`". In addition, the latter collection has a search filter associated with it. Note that Slivers may be attached to collections as well as items.

[XML]

```
<state>
  <collectionReferences>
    <collectionReference
      resourceId="highlights">
      <itemReferences>
        <itemReference itemId="H-1">
          <appearance displayMode="hidden"/>
        </itemReference>
      </itemReferences>
    </collectionReference>
    <collectionReference
      resourceId="source-1">
      <filter>
        <simpleSearch query="parts"/>
      </filter>
      <itemReferences>
        <itemReference itemId="EA-1">
          <appearance
            displayMode="expanded"/>
          <animation offset="0.5"
            duration="2.0" play="true"/>
        </itemReference>
      </itemReferences>
    </collectionReference>
  </collectionReferences>
</state>
```

[JSON]

```
{
  "state": {
    "collectionReferences": {
      "highlights": {
        "itemReferences": {
          "H-1": {
            "appearance": {
              "displayMode": "hidden"
            }
          }
        }
      },
      "source-1": {
        "filter": {
          "simpleSearch": {
            "query": "parts"
          }
        },
        "itemReferences": {
          "EA-1": {
            "appearance": {
              "displayMode": "expanded"
            },
            "animation": {
              "offset": 0.5,
              "duration": 2.0,
              "play": true
            }
```

```
              }
            }
          }
        }
      }
    }
```

A primary use of `collectionReferences` is to define the visibility and selection state of objects embedded *within* an experience—for example, a "pushpin" embedded in a Deep Zoom image such as shown in Figure 12. The `itemId` attribute identifies the specific item ("EA-1" is the ID of the left-most item embedded in the figure). The Sliver properties indicate that this item should change to the "`expanded`" state and (if possible) start playing any associated animation or video from a starting offset of half a second for a duration of two seconds.

The `collectionReferences` syntax applies to *any* experience that supports addressable embedded content. In fact this is leveraged to represent the Small State of an instance of the RIN player itself, as explained in Section 5.5. RIN narratives can thus keyframe not just the path through visually immersive content, but also control objects embedded within it, declaratively as instances of Small State present in logical keyframes within an Experience Stream.

Experience Providers are responsible for manipulating the state of embedded content based on the content of Experience Streams. Our RIN implementation provides a set of helper libraries to support keyframing of embedded content and these libraries are used by Experience Providers for a variety of media including panoramas, maps, paginated documents and even video.

Slivers have been defined for several additional targets and as the project evolves these definitions are being refined to cover as many cases with as few definitions as possible.

The user-manipulatable and scriptable aspect of experience state are represented by standard Sliver definitions where possible. Additionally, custom state Slivers may be defined for aspects of logical state of the user experience that are not covered by existing Sliver definitions. For example, a custom Small State definition is used for manipulating the interactive software programming experience described in Figure 1 (d). The custom state contains snippets of F# (programming language) code representing the evolution of small programs to convey programming concepts.

Small State may be used to both characterize and manipulate the real-time state of an experience. This may be used in a number of scenarios. A core use for Small State is to script animated paths

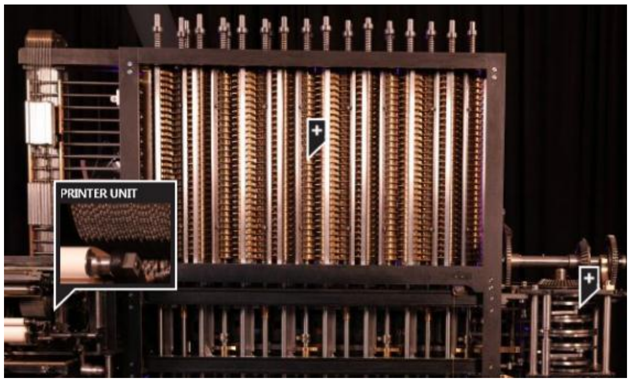


Figure 12: Illustrating Embedded Artifact state. The artifact on the left ("PRINTER UNIT") has ID "EA-1" and is in an "expanded" state.

through interactive experiences using the Experience Stream abstraction.

5.2 Experience and Experience Stream Structure

The Experience structure contains the information required by an Experience Provider to instantiate a particular rich media experience, and is also a container for Experience Streams associated with the experience. The specification of Experience is straightforward, as shown in the example below, with details elided for brevity:

[XML]

```
<experience id="map-1"
  providerId="CompanyX.MapProvider">
  <data>
    (experience metadata)
  </data>
  <experienceStreams>
    <experienceStream id="ES-1">
      <keyframes>
        <keyframe offset="0.0">
          <state>
            (Slivers)
          </state>
        </keyframe>
        <keyframe offset="2.0">
          <state>
            (Slivers)
          </state>
        </keyframe>
        (additional keyframes)
      </keyframes>
    </experienceStream>
    (additional Experience Streams)
  </experienceStreams>
</experience>
```

[JSON]

```
{
  "map-1": {
    "providerId": "CompanyX.MapProvider",
    "data": {
      (experience metadata)
    },
    "experienceStreams": {
      "ES-1": {
        "keyframes": [
          {
            "offset": 0.0,
            "state": { (Slivers) }
          },
          {
            "offset": 2.0,
            "state": { (Slivers) }
          },
          (additional keyframes)
        ]
      },
      (additional Experience Streams)
    }
  }
}
```

The `experience` structure encapsulates multiple Experience Streams, i.e., instances of *s*(*t*), each with a unique identifier. Each stream represents a labeled thread of exploration through the same experience. These labeled threads may be referenced from multiple Screenplays, providing the base capability to support nonlinear narratives. For example, Figure 1 (b) is a screenshot of a nonlinear narrative composed of narrative threads that are triggered by the user selecting an embedded object. This is implemented by binding a different Screenplay to the behavior attached to each embedded object. Clicking on an object causes the Orchestrator to start the corresponding Screenplay. The Screenplay in turn will reference the appropriate labeled Experience Streams from each of the Experiences that make up the composite experience. The result is a fluid transition to a separate narrative thread.

5.2.1 *From Keyframes to s(t)*

The keyframes represent partial discrete snapshots of Small State. The snapshots are partial in a very specific and constrained way we explain later. The `offset` attribute specifies the time offset from the start of the Experience Stream, thus a keyframe with time offset T represents a subset of Small State at time T.

Experience Providers are responsible for using these partial discrete snapshots to build a continuous "play" experience representing *s*(*t*). This requires an interpolation scheme that works with the partial nature of the keyframes. Each keyframe contains a subset of Small State Sliver targets. Each target, if present, must contain *a fully specified set* of properties for that Sliver. This constraint provides clear semantics and a practical way to interpret partial keyframes. To interpolate among a set of partial keyframes, the Provider must form independent "sub streams" of keyframes, one for each Sliver target. Each sub stream is a sequence of fully specified snapshots of a *particular* Sliver target (such as `viewport`.)

Some Slivers contain properties that are inherently continuous, requiring smooth interpolation, for example the `region` property. Other Sliver properties are discrete, such as `displayMode`. It is up to each provider to correctly interpret and interpolate the different properties that in aggregate make up Small State.

If a particular Sliver property is unspecified it is to be interpreted as taking on a default value. Thus an "empty" Sliver consisting of a target with no properties indicates a reset to all default values for that Sliver.

5.3 Screenplay Structure

The Screenplay structure is not part of the core RIN specification. Instead RIN supports pluggable Screenplay Interpreters, each of which may use a custom Screenplay format. This is primarily because this aspect of RIN is most in flux, and there is a possibility that we may use an existing standard, such as the SMIL 3.0 Timing and Synchronization profile [4], as well as the possibility that certain scenarios may call for more specialized, sophisticated and/or dynamic notions of a timeline. The current version (illustrated in Figure 7) represents a Screenplay as a simple list of Experience Stream Identifiers with associated properties that include the start offset, duration and optional z-Index. A sample Screenplay is presented below.

[XML]

```
<screenplays>
  <screenplay id="SCP-1">
    <data>
      <experienceStreamReferences>
        <experienceStreamReference
          experienceId="E-1"
          experienceStreamId="ES-1"
          begin="44.2"
          duration="8.1"/>
        <experienceStreamReference
          experienceId="E-2"
          experienceStreamId="ES-2"
          begin="52.3"
          duration="6.5"/>
      </experienceStreamReferences>
    </data>
  </screenplay>
</screenplays>
```

[JSON]

```
{
  "SCP-1": {
    "data": {
      "experienceStreamReferences": [
        {
          "experienceId": "E-1",
          "experienceStreamId": "ES-1",
          "begin": 44.2,
          "duration": 8.1
        },
        {
          "experienceId": "E-2",
          "experienceStreamId": "ES-2",
          "begin": 52.3,
          "duration": 6.5
        }
      ]
    }
  }
}
```

5.4 Segment Structure

The Segment is the top-level RIN construct. A sample Segment is presented below. Some fields are elided for brevity. In particular, not shown are metadata that can be inserted into the key structures (`segment`, `resource`, `experience`, `screenplay`, and also `experienceStream` and `keyframe`) by including a `data` element, which is an open-ended dictionary of objects. (We have seen one example of this in the Screenplay sample in the previous section.)

[XML]

```
<segment estimatedDuration="58.8"
         defaultScreenplayId="SCP-1"
         theme="modern">
  <providers>
    <provider
      id="mapProvider"
      name="Microsoft.Research.MapProvider"
      version="1.0"/>
  </providers>
  <resources>
    <resource id="R-1" uriReference="…"/>
  </resources>
  <experiences>
    <experience id="ES-1">
      …
    </experience>
  </experiences>
  <screenplays>
    <screenplay id="SCP-1">
      …
    </screenplay>
  </screenplays>
</segment>
```

[JSON]

```
{
  "estimatedDuration": 58.8,
  "defaultScreenplayId": "SCP-1",
  "theme": "modern",
  "providers": {
    "mapProvider": {
      "name": "Microsoft.Research…"
      "version": "1.0"
    }
  },
  "resources": {
    "R-1": {
      "uriReference": "…"
    }
  },
  "experiences": {
    "ES-1": {
      …
    }
  },
  "screenplays": {
    "SCP-1": {
      …
    }
  }
}
```

5.5 Deep-referencing RIN Content

The mechanism of Small State provides a general way to reference deep into RIN content. Conceptually, one can think of this reference as a Small State representation of the entire RIN player after the user has, through exploration, manipulated the composite experience to a particular state. Examples include:

- jumping to a particular time offset within a particular Screenplay;
- highlighting an embedded object overlaid over a map while viewed from a particular zoom level; and
- zooming into and highlighting a small section of a video, perhaps accompanied by a text annotation.

Deep referencing enables many scenarios including indexing and search into rich media, and user-created annotations over rich media.

5.5.1 *Small State Representation*

The state of a RIN player bound to a particular RIN Segment is represented using the previously-introduced `document`, `animation` and (optionally) `collectionReferences` Small State Slivers. The `document` Sliver encodes the Screenplay ID. The `animation` Sliver encodes the time offset along the Screenplay timeline. This highly concise representation is sufficient to encode the state of the player at a particular point in time along a particular Screenplay timeline. `CollectionReferences` encodes any *deviations* from the path specified in the timeline. More specifically, `collectionReferences` encodes `the` Small State of those experiences (if any) whose states *differ* from their implicit state at the specified time offset along the timeline. A sample is presented below:

[XML]

```
<state>
  <document screenplayId="SCP-1"/>
  <animation begin="2.0"/>
  <collectionReferences>
    <collectionReference
      resourceId="experiences">
      <itemReferences>
        <itemReference itemId="ES-1">
          (Slivers representing changes in ES-1)
        </itemReference>
      </itemReferences>
    </collectionReference>
  </collectionReferences>
</state>
```

[JSON]

```
{
  "state": {
    "document": {
      "screenplayId": "SCP-1"
    },
    "animation": {
      "begin": 2.0
    },
    "collectionReferences": {
      "experiences": {
        "itemReferences": {
          "ES-1": {
            (Slivers representing changes in ES-1)
          }
        }
      }
    }
  }
}
```

5.5.2 *URL Representation*

A deep reference into RIN content can be represented using a URL with the following syntax:

`rin://`*segment-reference*`?screenplayId=`*value*
`&begin=`*value*`#`*rest-of-state*

The fragment portion of the URL denoted by "*rest-of-state*" is populated with a stripped-down version of the JSON representation of the *content* of `itemReferences`, with whitespace removed and special characters appropriately escaped.

Here is an example (without escaping special characters):

```
rin://sample.xrin?screenplayId=SCP-1
&begin=2.0#{"ES-1":{…},…}
```

The use of compression to further reduce URL size is being explored. Note that the burden of parsing JSON is placed on the player. We chose JSON rather than inventing a new format for the serialization of Small State. Very simple players can simply ignore text after the "#" character, thereby limiting themselves to only referencing content along a timeline.

6. CONSTRUCTING AND TRANSFORMING CONTENT

RIN defines a document format and reference player architecture that dynamically pulls in Experience Provider libraries as required. There is no restriction on how RIN content is created and over time we expect an ecosystem of tools and services, including run-time services that may dynamically create content analogous to dynamically-created HTML, now pervasive on the Web. We have explored two ways of constructing RIN content that represent two ends of the spectrum. Our biggest investment is in RIN Studio (Figure 2) an Experience Stream and Screenplay authoring GUI tool that enables the composition of linear sequences of Experience Streams. This tool is a workhorse and was used to create the content in all the examples shown in Figure 1. We have also built a service, called DRIN (for Dynamic RIN) that *dynamically* creates simple RIN narratives on any Wikipedia topic. Given just a Wikipedia topic as input, DRIN does some simple structure analysis and crafts a single Screenplay and a set of Experience Streams that include a synthesized audio narrative and music track, as well as pan-and-zoom slideshow of images obtained by an Internet image search.

RIN content may also be transformed. For example, a more sophisticated Experience Stream can be transformed into a simpler one that loses functionality but has fewer run-time requirements and better archival characteristics. We call such transformations that convert one Experience Stream to another *Experience Stream Projections*. One concrete example that we have implemented is pre-rendering Experience Streams to video and having both the original and video versions available at rendering time. This allows for pre-rendered smooth interpolation when the narrative is playing, but when the narrative pauses, the original experience is brought up, properly synchronized (by supplying the current Small State) and ready to interact with the user.

The RIN format may even be used as a pure data representation with no intent of playing the content within a RIN player. For example, it may be used as a unified way to represents annotations over video, images, panoramas and maps. Another example is to use Small State as a general mechanism to convey state changes among cooperating subsystems. These are areas of current research.

Having defined the declarative semantics of RIN content, defined a set of standard Small State Slivers that define the logical state of a wide variety of experiences, and built a reference player and a stable of Experience Providers for these experiences, RIN is now at the stage where we are starting to work with a few collaborators on vertical scenarios that range from museum kiosks to rich websites. We feel that the full impact of RIN will be realized when the specification and reference implementations become used as the foundation of a broad range of scenarios driven by different organizations, with the RIN specification being the common lingua franca for interactive narratives over rich media.

7. RELATED WORK

There is a substantial body of work on interactive storytelling and nonlinear narratives. This includes general principles such as defined by Szilas and Rety [16], specific techniques for automatic and semiautomatic narrative construction, such as Raconteur by Chi and Lieberman [5], and frameworks for construction and authoring in vertical domains such as GeoTime by Eccles et al [6]. We consider RIN as potentially an enabling platform for many of these techniques, while broadening the scope of storytelling to include fluid and cinematic narratives over new kinds of rich media.

World Wide Telescope [11] is the pioneer of combining narrative (in the form of tours) with exploration of a large and complex world (the universe). The concept of combining tours with exploration has also been adopted by Google Earth [8], and most recently, by LADS [3] and ChronoZoom [1]. Each of these is a closed system that includes a built-in definition of the “world” over which the tours take place, and also the representation of the authored content is neither extensible nor designed to be authored or rendered by entities beyond a single proprietary system. RIN can be thought of a generalization of the concept of tours to cover arbitrary worlds, while defining the semantics of the resulting content so as to open up authoring and presentation. In fact, the creators of the LADS technology are exploring the use of the RIN HTML and JavaScript stack for the next generation of their software, called Touch Art Gallery (TAG).

There is an interesting body of work around storytelling in the context of data visualization. The GeoTime system by Eccles et al [6], while scoped to the vertical domain of stories over geo-temporal data (data with location and time attributes), describes a general system for obtaining “snapshots” of visualization state in their Capture Management System, which they describe as being capable of handling the states of other visualizations through a plug-in system. The actual data format is not described, and the captured snapshots are stored in a database in a proprietary format. GeoTime snapshots can be thought of as an opaque form of Small State. The concept of visualization state is also utilized in the Tableau system by Segel and Heer [15]. In both GeoTime and Tableu, states are discrete snapshots, and various points in the story can bring up the visualizations synchronized to particular states. These systems do not have RIN’s cinematic notion of Experience Streams that encode fluid paths through arbitrary worlds, nor do they compose multiple experiences into a holistic experience. The authors believe that there is interesting research to be done in combining RIN’s concept of Experience Streams, as well as the diversity of rich media that it supports with Tableau’s state-based approach to data visualization, to form rich narrative experiences that are both cinematically compelling and information-rich.

The Synchronized Multimedia Integration Language (SMIL) [4] is a declarative specification for multimedia and a W3C recommendation. The latest (3.0) version is split into several modules that include transition effects, animation, and timing. The bulk of the specification defines visual and audio effects at a far lower level than RIN. In fact, Experience Providers could well use SMIL as part of their implementation. In practice, however, combinations of CSS and JavaScript (or proprietary equivalents) are often used to implement rich media experiences. The part of SMIL that remains most relevant to RIN is the Timing and Synchronization module. RIN Screenplays allow pluggable Screenplay interpreters, and while our current implementation uses a relatively simple format, it is quite conceivable that more sophisticated timings can be encoded using SMIL timing and synchronization primitives with a corresponding RIN Screenplay interpreter. We have not felt the need for this, however, because RIN Screenplays orchestrate events at a much higher level, leaving lower-level timing and synchronization details to the Experience Providers.

8. CONCLUSION

Document standards are falling behind the capabilities of rich media experiences on the Web, with the dominant document standards (HTML and CSS) defined at a fundamentally low level. As a result, the higher-level functionality required to implement rich media experiences is frequently implemented as opaque, organically evolving, and often proprietary JavaScript or OS-specific libraries. End-to-end experiences are built as web sites with opaque back ends or as monolithic applications targeting the various application stores. The ability to mix and match and thread through rich media from multiple sources is practically nonexistent.

We introduce RIN, a presentation infrastructure driven by a high-level declarative specification that has a generalized notion of serialized logical state, called Small State, at its core. We define the runtime execution model, which may target any presentation platform, including HTML5 and JavaScript, and which processes declarative content that is defined at a logical level. Our initial focus has been on enabling a document-centric representation of interactive narratives that thread through an extensible set of rich media experiences. We feel that the concept of Small State could be used in a broader set of scenarios, such as collaborative editing and sharing of content—including users' exploratory paths through rich media—as well as use of Small State to characterize and analyze user interactions with rich media experiences. These are promising areas for future work.

We believe that RIN has the potential to enable a vibrant ecosystem of tools and services to create, analyze, transform, remix and present interactive narratives, bucking the current trend toward a Balkanization of rich media. To this end, we are presently building an HTML5 and JavaScript implementation of RIN, and at the same time refining the definitions of a standard set of State Slivers. Our goal is to provide a standards-based implementation of RIN to the broader multimedia community in academia and industry.

9. ACKNOWLEDGEMENTS

We would like to thank Sriram Rajamani for many stimulating discussions leading to a refinement of concepts around Small State. We would like to thank and acknowledged Curtis Wong for his pioneering work on combining narrative with interactive experiences; in many ways RIN can be thought of as a generalization of World Wide Telescope and earlier projects.

We would like to thank P. Anandan, B. Ashok, Gordon Bell, Michael Cooper, Steve Drucker, Anoop Gupta, Rick Szeliski, Andries van Dam, Matt Uyttendaele, and Curtis Wong for their long term advice and support that have helped shape the direction of this project.

We would like to acknowledge the past and present members of the extended RIN Team, responsible for building the RIN Studio authoring tool, the RIN Silverlight and HTML players and Experience Providers, hosting infrastructure, narrative construction and academic outreach: Nikhil Arjunagi, Joybroto Banerjee, Nikhil Chandran, Aldo John, Tanuja Joshi, Vinay Krishnaswami, Saurabh Kothari, Chinmay Kulkarni, Ajay Manchapelli, Gautham Mudambi, Prasad Naldurg, Vidya Natampally, Siddharth Prakash, Archana Prasad, Poornima Rajagopal, Kanchen Rajanna, Satish Sangameswaran, Aditya Sankar, Jatin Shah, Gopal Srinivasa, Anand Tekaday, Sridhar Vedantham and Sujit Warrier.

We would like to thank Ryan Lester for being an early user of the nascent RIN HTML and JavaScript stack.

This project was made possible thanks to the support of Microsoft Research Connections, the MSR India Advanced Development Group, and the MSR Redmond Interactive Visual Media Group.

10. REFERENCES

[1] U. C. Berkeley and Microsoft Research. *ChronoZoom*. Web site. http://eps.berkeley.edu/~saekow/chronozoom/

[2] Bostock, M., Ogievetsky, V. and Heer, J. D3: Data-Driven Documents. In Proc. IEEE InfoVis, 2011.

[3] Brown University. The Large Artwork Display on the Surface (LADS) project. http://cs.brown.edu/research/lads/.

[4] Bulterman, D. and Rutledge, L. *SMIL 3.0: Flexible Multimedia for Web, Mobile Devices and Daisy Talking Books*. Springer; 2nd ed. 2008.

[5] Chi, P.-Y. and Lieberman, H. Raconteur: From Intent to Stories. In *Proc. ACM IUI*. 2010.

[6] Eccles, R., Kapler, T., Harper, R., and Wright, W. Stories in Geo Time. In *Proc. IEEE VAST*. 2007.

[7] Element Collection, Inc. *The Elements: A Visual Exploration*. Apple iPad application. http://itunes.apple.com/us/app/elements-visual-exploration/id364147847?mt=8.

[8] Google. *Google Earth*. Application and Web Experience. www.earth.google.com.

[9] Gray, J., Szalay, A., (2002) The World-Wide Telescope, an Archetype for Online Science. *Microsoft Research Technical Report MSR-TR-2002-75*, 2002

[10] Kopf, J., Uyttendaele, M., Deussen, O., Cohen, M., Capturing and Viewing Gigapixel Images. In *Proc. ACM SIGGRAPH*, 2007

[11] Microsoft Research. *World Wide Telescope*. Application and Web Experience.

[12] National Film Board, Canada. *Bear 71*. Interactive Documentary. http://bear71.nfb.ca/#/bear71.

[13] Push Pop Press. *Our Choice*. Apple iPad application. http://pushpoppress.com/ourchoice/.

[14] http://www.worldwidetelescope.org/Home.aspx

[15] Segel, E and Heer, J. Narrative Visualization: Telling Stories with Data. In *Proc. IEEE InfoVis*. 2010.

[16] Szilas, N, and Rety, J-H. Minimal Structures for Stories. In *Proc. ACM Workshop on Story Representation, Mechanism and Context*. 2004

[17] Microsoft Corp's Deep Zoom. *Wikipedia article* http://en.wikipedia.org/wiki/DeepZoom.

[18] Microsoft Corp's Pivot Viewer. *Wikipedia article* http://en.wikipedia.org/wiki/Microsoft_Live_Labs_Pivot