Design Specification of an Animation System for Web Applications

Daniel Dyrda
Fakultät für Informatik
Technische Universität München
Garching bei München, Bavaria, 85748
Germany
Email: dyrda@in.tum.de

Animations have become an integral part of web applications. The dynamic context of interactive web applications leads to special demands on animations. Besides the existing animation frameworks for web applications, which are more concerned with the application of animations, our animation system focuses on the dynamic control and sequencing of animations at runtime.
Our system follows a property-driven approach that uses Animation Controllers based on Action State Machines which behave according to the internal state of the application.
We provide a design specification af an animation system for web applications which considers proven concepts of established animation systems used in interactive contexts such as game engines in addition to basic requirements of animations and their use cases in web applications.
Our concept facilitates the easy use and control of animations in web applications.

1 Intro

Animation is the illusion of motion created by displaying a sequence of still images. [2, 25]. It facilitates the understanding of process and continuity of an object. [1, 3]

Animations have always been a crucial component of various media. They are an integral part of films and computer graphics. [2] Especially in digital advertising they are used very consciously. [14] For some time now, they have been increasingly used in digital interfaces and web applications. [2, 15]

This trend can be attributed to various reasons.
Besides to some drawbacks of animations (see section 2.2.2), the advantages for their use in interactive applications are predominant (see section 2.2.1).
In addition, the ever-increasing performance of end devices makes it possible to use animations despite their real-time requirements and high performance demands. [15]

Nevertheless, there are some challenges to be overcome in order to use animations in interactive scenarios such as web applications. Animations add complex, dynamic aspects to an application. Predefined cases are not enough to meet these demands. A system for controlling competing animations is required. Animations must be dynamically sequenced based on unexpected, spontaneous events. [15]

Several application areas that have been using animations in an interactive context for quite some time, such as computer games, use animation controllers for these tasks. An animation controller facilitates the sequencing of animations in a dynamic and interactive environment. [1]

Such a system is also required for interactive web applications with dynamic animations. In this paper, we discuss a design specification for a system for controlling animations in web applications.

2 Related Work

There are many topics relevant to our animation system. The most important related areas are animation in a general term, the use of them in interfaces & web applications and animation frameworks in other interactive applications such as video games.

2.1 Animation

A general idea of animation as the illusion of motion is conveyed by many scientific papers [3, 8, 20].
In addition, an understanding of animation from a more creative point of view as described by Thomas and Johnston [21] is recommended.
When it comes to linking functionality of programs with animations, it is essential to understand the needs and requirements of creative people for their animation systems so that they can create the best possible outcome.

2.2 Animation in Interfaces and Web Applications

Besides to the actual understanding of animation, it is advantageous to know general classification and animation principles for interfaces as described by Baecker and Small [22].

When it comes to animation in web application in particular a more in-depth knowledge about web animation formats, the technologies used and the animation production process is essential.
Animation for web applications is a rapidly evolving field which is characterized by a high number of emerging technologies and practices.
Although it is desirable to have an understanding of these emerging technologies, this area is only partially relevant. A complete coverage of the technologies is desirable but almost impossible to achieve, if all technologies are considered separately.
Fortunately, almost all new technologies are more or less in line with the browser standards and conditions set by the w3 committee and other influential organizations.
For this reason, it is more important to know the basic standards and guidelines than to know individual, sometimes very short-lived technologies in detail. Current standards can be found in the w3c standards guidelines [24].

It is also advantageous to know similar web animation frameworks and libraries that are designed to provide higher-level interfaces to the application layer.
Related systems with interesting implementation approaches and problem-solving strategies such as the planning based animation control framework Player [1] or the visual programming model HandMove [15] can help to improve your own approaches.

2.2.1 Advantages

It is interesting to see to what extent animations improve an application and what kind of tradeoffs can be expected.
Researchers show that objects in motion automatically attract people's attention, cause a physiological arousal and increase the level of engagement. [4, 6, 9, 14]
Animated content appreciably improves the graphical perception. [4] It can help to understand changes in the user interface [8, 10, 11] and help the users to keep their orientation. [5, 6]. Furthermore, it supports the understanding of underlying data [2, 3], learning processes [3] and decision making [7].

2.2.2 Disadvantages

However, negative consequences can also occur as a result of excessive or incorrect use.
Some researchers assume that animated objects require a higher cognitive load than static ones. In some situations with many animated objects this can lead to a mental overload of the user, so that she can no longer process the displayed information adequately [2, 8, 12–14, 23].
Another drawback is that animations are played back over time. Thus animations inevitably delay the visual feedback of an event. This results in an inadvertent idle time for the user. [3, 4, 8]

2.3 Animation in Video Games

Video games are interactive applications with similar requirements and areas of application to web applications.
Animations have long been an integral part of video games. This has resulted in animation systems that have proven themselves over many years and in countless projects. Knowledge of these established systems is recommended to find optimal solutions for web applications.
Gregory provides a comprehensive overview of common requirements and implementation standards of animation frameworks in real-time computer game engines in his book "Game Engine Architecture" [25].
Proven animation control systems are used in state-of-the-art 3D game engines like the Unity3D Engine [26] and the Unreal Engine [27].

3 Animations in Web Applications

Animations in web applications emerge in various forms. Basic elements of web applications can be animated using Cascading Style Sheets and JavaScript. Many APIs enable easy and efficient animation of objects. Complex aspects can be animated with more complex systems such as WebGL. In addition, animations can also appear in the displayed content such as embedded videos.

Before we take a closer look at animations and the technical aspects, we should study their use in web applications. If we understand how they are used in web applications, we can optimize our system to these situations and ensure that the app developers can easily implement these situations.

Web applications are interactive real-time applications that require sensitive memory management and are highly dependent on performance.
Interactivity means inherently that the relevance of the interface between application and user is particularly high.

Web applications as an environment has significant effects on animations. They must meet the technical requirements. Animation controllers are necessary to cope with the interactive character if more complex systems are to be implemented.

3.1 Use Cases of Animations

There are many different reasons why animations can be built into an interface or application. The following list shows meaningful use cases, what intentions they are pursuing and examples.

This overview of common use cases for animations and the intentions are from Baecker and Small [22].

Identification *What is this?* An animated lock that indicates that a text input field is intended for a password.

Transition *From where have I come, to where have I gone?* Scrolling animation from a section to another after clicking an anchor link to a position on the same page.

Choice *What can I do now?* Grey out buttons to indicate that they are no longer active/accessible.

Demonstration *What can I do with this?* Text which appears letter by letter to indicate that text can be entered here.

Explanation *How do I do this?* A mouse icon with an animated mouse wheel that indicates that this element contains scrollable content that can be explored using the mouse wheel.

Feedback *What is happening?* A progress bar that indicates the application is operating in the background or an error window popping up that indicates what has gone wrong.

History *What have I done?* Menu items that are fading out to show that these items have already been visited.

Guidance *What should I do now?* A flashing button that links to the next page.

Animations can also belong to several categories. A classic intro animation of a website can show that the page is being loaded (Feedback). It can also explain to the user which page she is currently visiting and which topics await her (Identification).

In addition to interfaces, animations can also appear in the content of the web applications. These are mostly interactive applications such as games or the visualization of data. Here, animations are often used to bring objects to life or to pursue storytelling goals of the content. The extent to which animations should be used, also in addition to the above-mentioned reasons, depends largely on the content, what is intentionally pursued in general and which format it is. This should be evaluated on a case-by-case basis.

Further uses of animations are conceivable, but not recommended. Animations that do not fall into one of the above categories such as emotional animations which do not have a deep intention, i. e. are only meant to be pretty or demonstrate a technique, are usually not purposeful and should be avoided.

3.2 Sources of Animation Intentions

A spontaneous animation intention can have various origins. It can be differentiated according to the content structure of the web application.

Web applications usually have a similar content structure. (See figure 1) Web application consists of many objects. Several objects that are functionally related to each other form widgets. Objects and widgets can form consistent elements, which are arranged in the application.
In addition, applications can usually be divided into sections. Sections combine elements that belong together in terms of

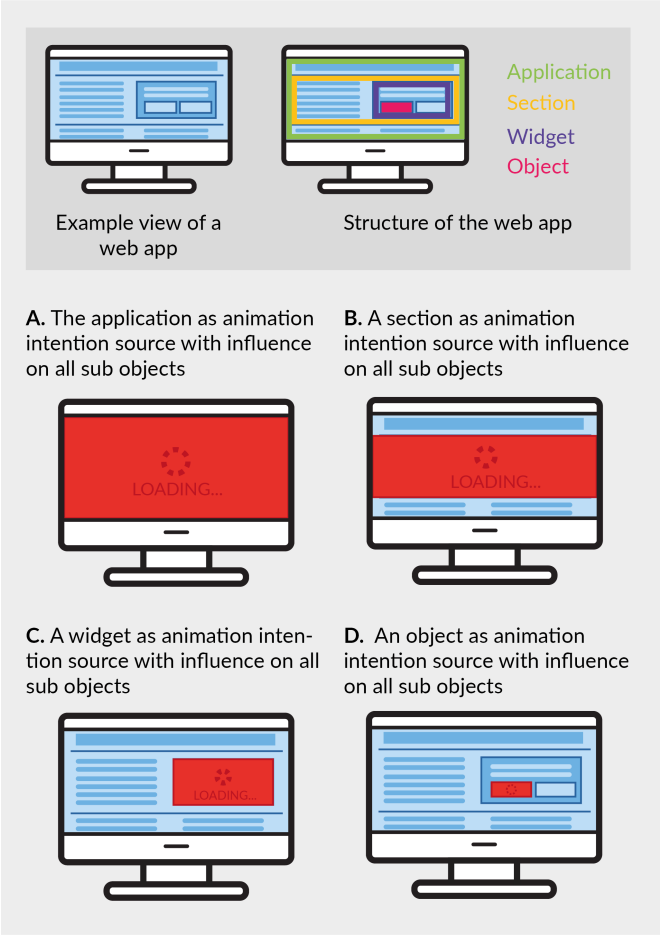

Fig. 1. Example web app with different animation intention sources and scopes.

subject and space. Following this, we get this structural hierarchy of common structural elements of web applications:

- Application
- Section
- Widget
- Object

An example of this structural break down is an interactive Portfolio (application) which offers a contact section (section) with a contact form (widget) which in turn contains a send button (object).

Due to the tree-like structure of web applications, which is defined by the Document Object Model (DOM) Structure, elements are always in a parent-child relationship to each other. Each element has exactly one parent element.

Each of these structural entities can be logically the origin of an animation intention. The scope of the intention depends on the hierarchically position of the affected objects. Usually those animation intentions would have at least an influence on all subordinate elements.
An example is shown in figure 1 on page 3: A loading animation intention of the web application would cause all elements to be affected by this intention and display a global

loading screen animation. However, if only one section is reloaded, the other sections can remain unaffected. The application would still work normally, only the elements of the loading section would show a loading animation.
Following this the loading intention of a widget would only affect the elements of this widget. An Object would only affect itself and other child objects.

3.3 Animation Trigger Agent

Various agents can trigger animations. In interactive web applications, there are two main players involved: The application itself and the user input.

Although generally the role of the trigger agents is only of limited importance, this is especially important for web applications. Many of the functionalities related to user input are provided by the browser. Therefore, the interaction between application and the browser plays an important role for dynamic animations.

4 Animations

In this section we will examine the structure of animations. Step by step we build up an overall pattern to get a general model of animations. This model is shown in figure 4 on page 6.
To do this, we look at the actual content of the animation, which defines the displayed behaviour and the hidden control parameters that are necessary to play animations dynamically.
We use this model to define the interface between our animation control system and external animation libraries.

First of all, we examine what kind of objects can be animated and which techniques are commonly used.

4.1 Animation Techniques

Animation is the illusion of motion.
The illusion of motion can be created by displaying a sequence of still images of an object. [2, 25]

In order to obtain a sequence of still images of an object, there are several common used animation techniques. Which technology is suitable depends on many factors. One factor is the format of the object.
There are two main representations of objects:

- **Images of objects as two dimensional array of pixels.** The object is already rendered and represented as a two dimensional array of colour information called pixels.
  Typical examples are traditional image and video formats. These formats are already available in a corresponding format and can be displayed easily. An example is shown in figure 2 A. on page 4.
- **Spatial representations of objects as set of vertices.** These representations are used in 2D and 3D real-time rendering engines. Three dimensional models often consists of meshes and materials.
  Typical examples are data formats such as the OBJ or FBX format for 3D models. There are also similar structures in 2D such as Scalable Vector Graphics (SVG).
  All formats in this section have in common that objects are represented by a set of spatial vertices with additional information. A renderer have to render the objects each frame in order to get an image which can be displayed to the user.
  An example is shown in figure 2 B. on page 4.

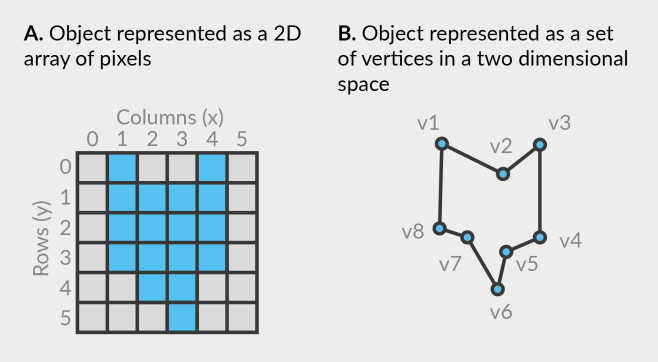

Fig. 2. Examples of two common object formats.

Based on these two basic formats, various animation techniques have evolved with different weightings between complexity and performance:

- Traditional hand drawn **Cel Animation** displays a sequence of hand drawn images in a rapid succession. [25] With the evolution of the computer, an electronic equivalent to Cel Animation, the **Sprite Animation**, has emerged.
  Displaying prerendered and hand made images can be very data intense and the possibilities of dynamic animations in a dynamic environment are limited. Especially interactive real-time applications with lots of different animations such as video games and web applications can reach their limits.
  These animation techniques are used for animated images.
- A simple way to animate spatial models is the **Per-Vertex Animation**. It provides high complexity and nearly unlimited possibilities for life-like animation creation.
  Each vertex of a 3d model (or similar data structure) is transformed individually each frame in order to simulate motion. This technique is very data intense. [25]
- A simpler and less compute-intensive solution is the **Rigid Hierarchical Animation**. [25] This approach works similar to the Per-Vertex Animation. The main difference is that not every vertex, but hierarchical collections of vertices called rigid bodies are animated. The result is that only groups of vertices need to be animated, not each vertex individually.
  In addition, the hierarchical structure of the rigid bodies

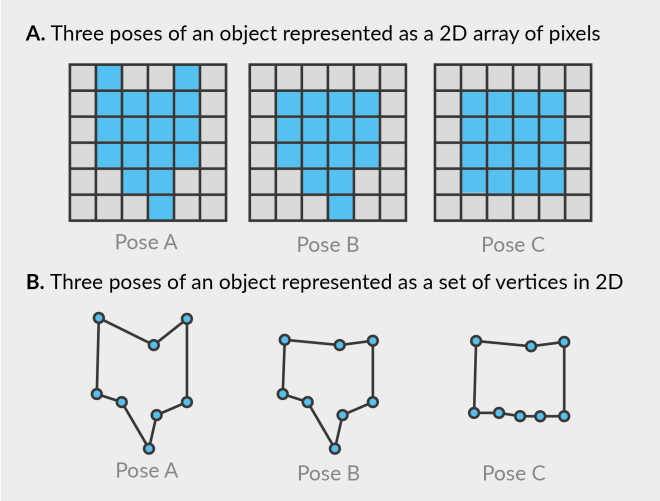

Fig. 3. Examples of three different poses of two common object formats.

influences each other, so that not even every group needs to be animated in order to achieve a harmonious animation. This significantly reduces the required amount of data and the computing effort.
- There are many other animation techniques. Many of them are based on the above-mentioned or are mixed forms, such as the Skinned Mesh animation technique.

4.2 A General Animation Model

Now we will take a closer look at the technical implementation of animations. Therefore, we analyze them and create a general model which is visualized in figure 4 on page 6.

The illusion of motion can be created by sequencing and visualizing discrete, still **Poses** of an object.
Poses are snapshots with concrete values of a property set that determine the appearance of an object at a certain point in time. These sets with concrete values could be:

- ...an image with a value for each pixel for sprite animations. An abstract example of poses of an object represented as a 2D array of pixels is shown in figure 3 A on page 5.
- ...a set of vertices with concrete transforms and colour properties for vector graphics or 3D models. An abstract example of poses of an object represented as a set of vertices in 2D is shown in figure 3 B on page 5.

Poses can be understood as basic elements of an animation.

A sequence of key poses on a local time line forms a **Clip**. Each property of the poses results in a property channel with a specific value at the respective points in time of the key poses.

The playback of a clip takes place over time. [3, 25] A Clip can be understood as a function returning a sampled property snapshot for a given time t. With T as the **Duration** of the clip, t=0 is the very first pose of the clip and t=T is the end of the clip.

In addition to the poses, the clip can be extended by **further channels**. Different types are conceivable. Simple values as well as channels with event calls and function calls are useful. These channels are used to implement functionalities which do not directly control the appearance of the object, but which must be synchronized with the animation.
For example, a channel can specify at which point in an animation the user's interactions should be ignored and when the user input should be able to change the state of the object. Another use case is calling up audio functions to add sound to the animation.
Simplified, a clip can be considered as a collection of time-varying parameters. [2]

In real-time applications the frame rate of the application depends on the processing power and program settings. This results in strong variations in frame rate. As a result, the poses of a clip are usually between the frames. Interpolation between successive key poses is necessary in order to provide pose sampling at any time t during the clip. [25]

A clip may contain an entire sequence of motions. It could play back the sequence of motions from the start of the application and continue running until it is finished or the user closes the application. Everything would be fine if the user just wants to consume a static animation.
However, one clip containing a set of motions is not suitable for interactive applications. Since the behaviour of the objects and the resulting sequence of motions depends on the user input, it is impossible to predict how objects will behave at runtime.
That is why one general pre-sequenced clip containing all movements does not make sense. Instead, a set of clips with individual movements is required. At runtime, the best matching clip can be selected from the set of clips and played back. This dynamic sequencing of animation clips at runtime depending on external properties is part of an animation system.
To enable sequencing and management, the clips must provide an interface for the dynamic playback. This interface can be understood as a **Wrapper** of the clip providing control parameters.

Additional time parameters are required to control clips. A **Timescale** property is the basis for playback. The Timescale determines whether a clip is played in slow motion or fast-forward, and whether it is played backwards. The timescale factor is taken into account when converting from query time t to the local time of the clip.
Another important factor for time related adaptations is the concept of **Easing Functions**. Easing functions provide time distortion effects. The result of the easing function is used to sample the corresponding pose.

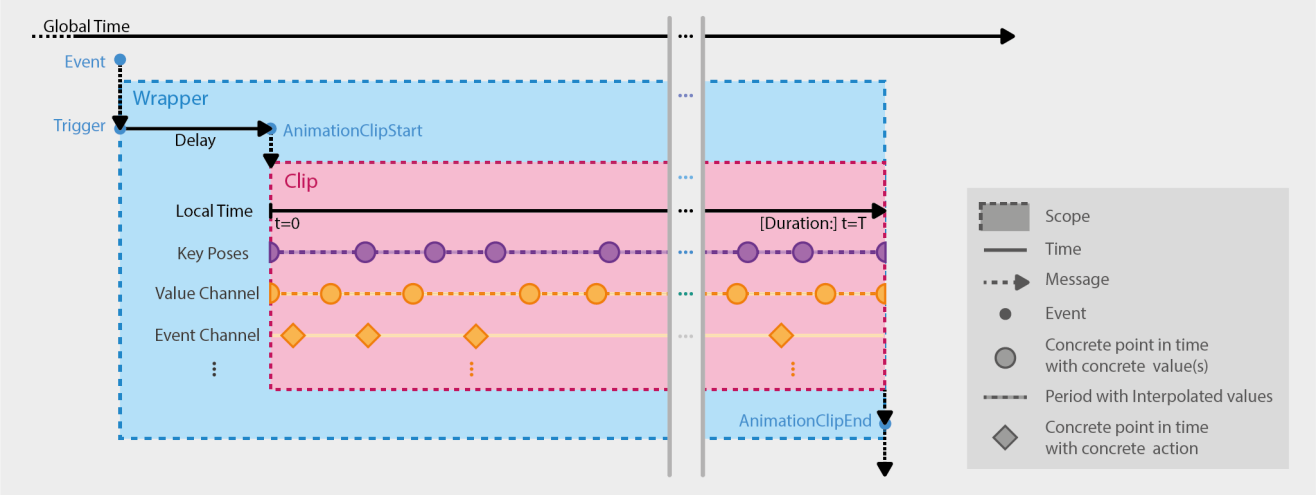

Fig. 4. A general animation model.

Furthermore it must be possible to **trigger** a clip. A **delay** property is necessary to determine the interdependencies between trigger and the actual playback start of the animation clip.

The completion of an animation clip should be a possible trigger as **call-through** for an animation clip. This enables short scope sequencing without further knowledge of the clip durations for specific triggers and animation loops.

5 Animation System Design

We have already seen how animations are used in web applications and how animations work per se. The next step is the quest of how animations can be sequenced and applied dynamically at runtime.

To this end, we define a system that corresponds to all the aspects mentioned above, implements common requirements and refers to existing systems for web applications. Our system implements proven concepts of established animation systems from the unity game engine [26] and the unreal game engine [27] as well as common game engine animation system aspects [25].

Animation systems are normally divided into three levels: [25, 26]

Animation Pipeline This level is deeply integrated into the system. For each animated element, the following steps are performed: Multiple clips are taken and blended using blend properties. The output pose is then calculated from the blend result. Subsequently, post-processing tasks are applied. [25]

Action State Machine This layer is on top of the Animation Pipeline. A finite state machine determines which clips are forwarded to the animation pipeline depending on the information from the Animation Controller. [25]

Animation Controller The Animation Controller is the high level interface between the application and the animation system. It is used to hide the presentation details from the application. [1, 25]

In web applications, the lowest layer, the animation pipeline, is mainly controlled by the web browser. Implementing this can be difficult and can have a strong impact on the performance of the application.
Our system is limited to the other two layers, which is responsible for a reasonable control of the animations. The actual application of the animations by the browser is not affected.
Complete blending between animations at the same time and at transitions is therefore difficult to realize, because this process occurs in the Animation Pipeline Layer. We will only offer a simplified approach to blending animations in state transitions.

5.1 The Idea

The basic idea of the system is best illustrated by an exemplary workflow. This process flow can be divided into two segments: Topics which are part of the development phase and the behaviour which are then executed at runtime of the application.

5.1.1 Development Phase

The following tasks are carried out during development. They only have to be done once and then remain constant for the runtime. The tasks already reflect the various components of the system.

Create the animations The developer implements Animation Objects for various elements of the application and for the required different states of these elements. These animations are created with respect to dynamic playback and sequencing from an external animation controller.

Define a general status description Later on, the objects should change their state according to the status of the

application and play the appropriate animations. To do this, the status of the application must be determined: the developer defines a model which describes the status of the applications, the sections, the widgets or the objects as required.

Define the animation controller properties Not every animated object or group of animated objects requires the same information about the status of the application. For each animation controller, the developer determines which information and properties are of interest to the controller. The developer also specifies where the information is to be obtained from if it has not been set directly.

Implement the action state machines The developer has already considered which states an object has in the first step. Now the developer implements the states in the Action State Machine. The states are also brought into relationship with each other. The transitions between states are based on the properties about the status of the application, which were defined in the previous step.
The corresponding Animation Objects created in the first step are referenced in the respective states of the Action State Machine.

Feed the animation controller with information At runtime, the animation controllers must be informed about the status by the application so that the action state machines can behave correctly. For this purpose, the values of the animation controller can be easily updated. The developer can simply integrate the updates into his normal program code or implement a new script, which is only responsible for the updates. Later on, the information will be continuously updated in the normal program flow.

5.1.2 Runtime Phase

The actions of the runtime phase are executed in a loop. It is not necessary that all aspects are executed at the same interval.
For example, the description can only be updated every few seconds if the status of the application does not change, while the animation controller changes the current state several times.

1. The program code updates the information about the status of the application.
2. The Animation Controllers update their description model of the current application status using the informations from the program code and their dependencies on other animation controllers.
3. The Action State Machine analyzes the current state of the machine. Based on the application state description of their animation controller, it may transition to a new state if necessary. [1]
4. The Animation Object of the current state of the Action State machine is resolved: The implemented behaviour of the Animation Objects which handles the animations are called up.
5. External visualizing scripts react to new classes or called methods adjust the visualizing objects directly: The elements of the web application are animated as expected.

5.2 Property Based Approach

An important concept for our system is the property-based approach of the system. The application does not specify the actions to be animated directly, but merely describes the state of the application. The animation controller behaves according to the description.

The great advantage of this is that the systems are even more separated from each other. The application can describe its status in a fire-and-forget manner and does not have to take care about the current state of the animation system or the results. The application is spared by Animation micro management. [25] Animation controllers can always behave as defined, regardless of when the current description of the application is from. The system can then react to the statuses if the internal processes make it possible. The system remains consistent and does not block. In the worst case, the system displays slightly past states.

5.3 Architecture

Based on our idea of the animation system, we get the following architecture for our system. Our architecture is based on the Seeheim Model. [1] An example overview is shown in figure 5. The system consists of 4 major components:

AnimationObjects This component contain the animation itself implemented by the app developer. It provides an interface for dynamic animation sequencing and controlling. Each consistent animation of a state is represented by an Animation Object.

Animation Controller Each animated element of the application, which later shows its own behaviour independent of other animated objects, needs its own animation controller. The animation controller facilitates the sequencing of animations in a dynamic and interactive environment. [1] This component consists of two major subsystems:

- **Application State Model** This subcomponent of an animation controller contains a model with properties which describes the status of the applications as required for each Animation Controller individually and the dependencies between the state models of the different Animation Controllers.
- **Action State Machine** Each Animation Controller has an Action State Machine. The Action State Machine defines the states of an object, the relationship between the different states of an object and how it will behave at runtime depending on the description of the application. [1]

Description Modules The Description Modules describe the state of the application using the model defined in the animation controllers by setting individual properties of

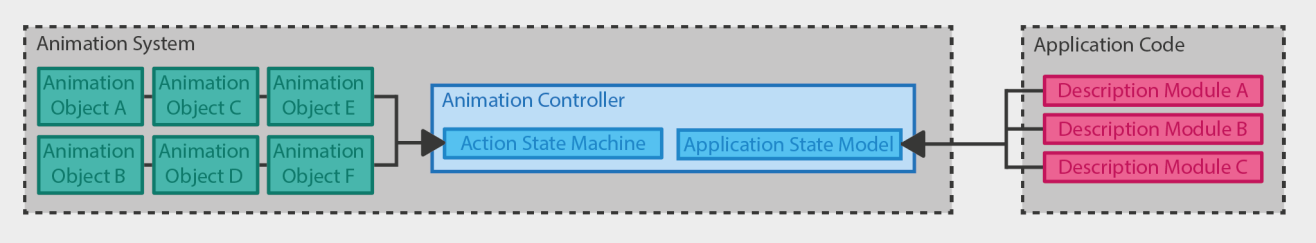

Fig. 5. A model of the architecture of the animation system.

the Animation Description Models of Animation Controllers. These modules are either sparsely integrated into the common application logic or can act collectively in an extra component. The implementation can therefore be adapted to the architecture of the application. They update the description of the status as required. This can vary greatly from property to property - it can be in certain situations only or every frame.

5.4 Performance

Our system primarily simplifies the application and handling of animation control. It offers the possibility to implement animations and their control in a structured way and to not have to worry about many needed features.
The system is always designed with a view to a high-performance conversion, because otherwise the actual application of the system would not be possible anyway. Once you have understood the system's behaviour, you still have to take care of the efficient and meaningful use of the animations, as it is usually the case.
For example, rewriting layout properties, such as setting the width or height of an element, can lead to expensive repainting of the whole website. This could lead to performance problems. [28] Our animation system can not optimize the animations implemented by the app developer or scale down the requirements. The conscious and considerate implementation of the animations for the system is therefore indispensable.

6 Animation Object

The Animation Object Design is based on the general animation model explained in section 4.2 on page 5.

Our Animation System provides Animation Objects which are designed as wrappers for animation behaviour defined in external animation libraries. The Animation Objects serve as an interface between our animation system and the application.
The animation system calls the desired behaviour on the Animation Objects. The application developer only has to implement the respective behaviour blocks of the Animation Objects.

6.1 Parameter

The parameters are grouped in an Animation Object Property Block with the following properties:

Delay The delay of the animation start in seconds
Duration The duration of the Animation Object in seconds
Timescale The time scale of the Animation Object
EasingFunction The easing functions of the Animation Object
CallThrough An Animation Object which will be invoked after this Animation Object has finished.
CallThroughProperties An Animation Object Property Block which will be used for the invocation of the call through Animation Object.

It is conceivable that each property of a clip can be passed with the invocation at runtime. If no values are passed or this is undesirable, default values of the clip can be used for calculations. If the clip has no default values defined the animation system should provide fall back values for such queries.

6.2 General Behaviour

As already mentioned, there are countless animation systems and frameworks for web applications. Almost all of them can be called and controlled within a JavaScript function. The behaviour of our Animation Object is based on this premise.

Our Animation Object is executed in at least two stages:

Start The first call occurs when the animation is started. The start up behaviour defined by the developer is executed. External animation libraries can be called depending on the properties defined in the Animation Object Property Block of the current Animation Object.
End A last call occurs after the end of the animation. The effects of the animation can be reworked. Call-throughs are executed afterwards.

6.3 Update Behaviour

In order to support as many types of animation systems as possible, we offer three modes how the system can behave during animation. These methods cover a wide range of animation libraries. Almost all types are supported, from a fire-and-forget manner to the manual resolving of the animation poses directly in the code.

UpdateType.Never The Animation Object methods are only called at start and end. There are no further calls during the animation. This method is preferable for animation techniques where the animation itself is initialized with a single call and the execution is then passed on to an external animation system.

UpdateType.Fixed The wrapper fixedUpdate method is called during execution in an arbitrary but fixed interval. This method is used for animation techniques which have to pass the execution of the animation intermittently to an external animation system, but always call new functions or validate the system state.

UpdateType.Frame The wrapper update method is called each frame. The implementation by the developer can set the values of the object depending on the current time of the playback each frame.

6.4 Support for Dynamic Transition Handling

Dynamic transitions between animations can occur at any time. It may happen that the animation is currently during execution. We provide two approaches to deal with such situations:

Abort Function An abort method should abort the animation started in the start method, so that the next animation activated by the animation system can work correctly.
The app developer must implement the abort function and reset all behaviours that have been initialized so far.

Consistent Proberties at State Machine Level Later on, a solution at the state machine level will be presented, based on animation systems that completely overwrite older animations when they set the same properties without being influenced by these older animations. Read more in the Consistent Property Principle subsection 9.1 of the Action State Machine section.

Which method is appropriate depends on the situation and must be decided by the app developer. Both methods can also be used in combination.

6.5 Animation Retargeting

A common requirement for animations is their reusability. Animations are created once and applied to different objects with similar structures. This process is called retargeting of animations. [25]

An example is a walking animation for a human being. The animation is created on the human model. This animation should then usually be applied to any other human model that has a body, a head, two arms and two legs. The same walking animation for a human being can also be applied to a slim elven or bulky orc. So a lot of work can be saved and similar animations do not have to be created again and again.

Based on the Document Object Model and classes, the system of web applications makes it easy to assign targets for animation using the classes or similar concepts.

An example: The close button of a widget should be visible and selectable when the widget is opened.
The animation can be applied by animating the object with the class close-button which belongs to the widget. Where exactly the object is located in the structure is not relevant and can vary slightly from widget to widget. We can retarget our close button animation onto every widget which has a similar close button and uses the close-button class for it.

Our animation objects are created with this concept in mind. However, the final focus of the implementation on the concept of retargeting is part of the animation implementation itself. This must be taken into account when implementing the start and end behaviour as well as the update behaviour of the animation object by the app developer.

7 Animation Controller

The animation controller controls and sequences the animation clips of an animated element at runtime. The Animation Controller orchestrates two included subsystems: the Application State Model and the Action State Machine.

An Animation Controller has exactly one Application State Model component and several layers of Action State Machines. Each layer has an Action State Machine which behaves completely autonomously and independently of the other layers based on the information from the shared Application State Model.

The multiple layers offer an easy way to animate several similar sub-objects independently and still share the same Application State Model.

The Animation Controller serves as a facade for its subsystems. The application code is not affected by the animation micro management. This is hidden by the Animation Controller. [25]

8 Application State Model

The Application State Model is a collection of different State Properties of different data types. These State Properties serve as control parameters for the Action State Machine.
The Application State Model is based on the property based approach of the animation system (see section 5.2 on page 7).

The Application State Model represents the interface of the Animation Controller to the Description Modules as well as to the Action State Machine.
The Description Modules can provide the Application State Model with current information about the application, which

is then provided to the Action State Machine.

The model defines the parameters describing the application, which are interesting for this Animation Controller, and where this information is obtained from.
Current information can be set by Description Modules or obtained from other Application State Models of other Animation Controllers.

8.1 State Properties

State Properties are named key-value pairs. Each name is unique for each Application State Model and serves as identifier.
State Property values can have one of the following types:

- **Int** An integer number
- **Float** A floating point number
- **String** A string of characters
- **Bool** A truth value.
- **Action** An action acts as bool which causes only one transition. It is set to false after a transition or a timeout which is defined for this property.
- **Class/Subclass** The name of this value is used to check if a class with this name is present on the element or not.

8.2 State Property Dependencies

For each State Property, it can be determined how the value is updated. By default, it is assumed that the values are set by Description Modules.

It often happens that many animation controllers need the same properties of the application as parameters in their Action State Machines. It can be cumbersome to set this same value in all Animation Controllers independently.
State Properties can reference other Animation Controllers and get their value from this Animation Controller if the reference contains a state value with the same identifier.

The structure of the dependencies can correspond to the structure of the sources of animation intentions (see section 3.2 on page 3).

An example: A Description Module can set a widget to active because the widget's function is currently available from the widget's point of view. The entire section may be inactive at the same time, because a lightbox display is active.
It would be disadvantageous if the Description Module of the widget would have to check all the facts before it can set the values. Nevertheless, the widget should be inactive in this situation.
The property value *isActive* can be linked to the appropriate property of its parents by dependency. The widget will only be active if itself is active and all parents are active too.

These principles and the resulting functionalities are part of the Animation Controller Application State Model in order to keep the separation of the Animation Controller dependencies and the Description Modules.

8.2.1 Referencing

It is conceivable that several animation controllers could serve as a reference at the same time.
If the value is to be obtained from other animation controllers, there are different ways of referencing it.

Direct Reference a specific Animation Controller.
Multiple Direct Reference multiple specific Animation Controllers.
Level X Parent Reference the Xth parent Animation Controller of this controllers element.
Until Level X Parents Reference all parent Animation Controller of this controllers element until the Xth parent Animation Controller (that included).
Root Reference the highest Animation Controller in the parent tree of this controllers element.
Level X Children Reference all Xth child Animation Controllers of this controllers element.
Until Level X Children Reference all child Animation Controllers of this controllers element until the Xth child Animation Controllers (those included).

Finding the referenced Animation Controller can be precalculated and does not have to take place at runtime. Only the number of referenced animation controllers and how their values are combined has an effect on the performance at runtime.

8.2.2 Combining Multiple Values

If multiple animation controllers are referenced, depending on the type of the value, a method to combine all values has to be determined. The method has to take an array of values of the type of the values to combine and return one combined value. It is conceivable that predefined functions are offered for different types.
For example, the logical operators *And*, *Or*, *exclsuive Or*, etc. could be offered for types, which are based on booleans or algorithms such as *Average* , *Min*, *Max*, etc. for number types.

8.2.3 Linked Dependencies

To be able to precisely define the dependencies between two Application State Models, their relationship must be determined.

- Should the value of the referenced Application State Model always be transferred or only if the value was not set on the element itself.
- In addition, it must be determine how the values are transferred. Is the referenced value simply taken over directly, or, similar to combining multiple referenced values, combined with the original value in a certain way.

8.3 State Properties Update

The dependency of one State Property on another can be implemented in an observer and observable relationship. State Properties can be observer and observable.
This ensures that the dependent property can immediately accept or calculate the new value as soon as a property changes.

It is important to be careful not to create circles in the dependencies. A tool that recognizes and warns against such situations would be an advantage.

8.4 Setting State Properties

The Description Modules can interact with the Application State Model in a fire-and-forget manner.
Values can be set by calling a method of the animation system and specifying the Animation Controller, the identifier of the State Property and the new value.

9 Action State Machine

An object should be able to perform various actions in an application with different animations.
Action state machines provide an simple solution to control and sequence these actions and the corresponding animation clips. The solution is based on the assumption that an object is always in a certain state at any given time. If certain circumstances are accomplished, an object changes its state.

Simplified, a state machine is a graph consisting of nodes connected by edges. Nodes are certain states which define the animation of the object and edges are transitions between the nodes with prerequisites. Transitions are performed if the prerequisites are fulfilled.
Following this, an Action State Machine has always exactly one currently active state and thus an active animation.
An exemplary Action State Machine is shown in figure 6.

9.1 Consistent Properties Principle

Consistent properties at State Machine level is one approach to support flawless dynamic transition handling.

There are animation systems that simply overwrite previous definitions without previous calls remaining relevant for the appearance of an object.
In these cases, a new animation can be called up without having to actively cancel the old animation if all animated property values are overwritten by the new one.
This can be guaranteed if properties are defined at state machine level, which are then set by each animation referenced within the state machine.

If the consistent properties at state machine level approach is used, the consistency of the system should be automatically checked and warning messages issued in case of rule violation.

Write Default An Action State Machine can provide a method which sets the visual properties of all objects affected by this Action State Machine to default values.
This method must be implemented by the application developer and should be in line with the consistent properties principle.
It can be used in the scope of the entire Action State Machine to restore the default values if needed.

9.2 States

States are represented as nodes of the Action State Machine. Most states provide a final animation or pose. [27]
Special states that do not offer animation provide only functional solutions for the developer. They are treated as full-fledged states, except that no content has to be defined.
At runtime, they are never the active state. They are only used to extend the flow of the Action State Machine and their subsequent transitions are resolved immediately if they would become the currently active state.

There is always one current state in a proper Action State Machine. Following this, there is always an animation available for play back.

9.2.1 Common State

The common states form the basis of the Action State Machines. They fulfil the task of the States in the simplest form: they provide an animation.
A Common State have the following properties:

Animation Object The referenced Animation Object is the content of the state.

Loop The loop property determines whether the clip should be played in a loop or not.

Speed The speed property defines the speed of the animation playback. The duration of an animation playback of this state is the duration of the content multiplied by the inverse speed. For example a speed of 2.0 plays the clip twice as fast and a speed of 0.5 doubles the playback time of the clip.

Write Defaults This property defines whether the WriteDefaults method should be automatically called at the end of the animation playback or not.

Transition A state has incoming and outgoing transitions which describe how this state is linked to the rest of the state machine.

Exemplary Common States are shown in figure 6 on page 12.

9.2.2 Hierarchical Variation State

In many cases the appearance of objects and their animation should not change fundamentally, but should still react to modifications.
These requirements are similar to pseudo classes from CSS. The interaction by the user should only cause small adjustments of the animation, e.g. only increase the transparency

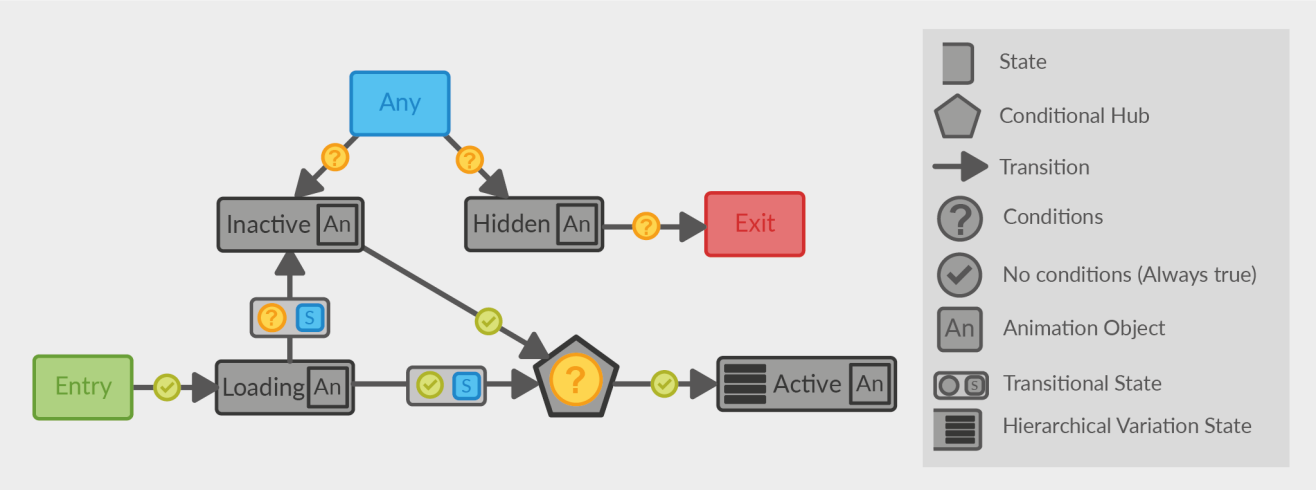

Fig. 6. An exemplary Action State Machine with three Custom States (Inactive, Hidden, Loading), one Hierarchical Variation State (Active) and several Transitional States. The relations between the states are defined by Transitions and Conditional Hubs.

of an object without redefining the appearance of the whole object.

In addition, it must be possible to arrange these adjustments hierarchically. Many modifications of the environment contain other modifications, but often only one customization should be applied.
For example, clicking on an object often also means a simultaneous hover interaction. In most cases, however, only the click animation should be played.

In common animation systems for other target platforms, this behaviour would be achieved with blend trees. Unfortunately, blend trees require a deep integration into the low-level layer of an animation system to be implemented efficiently. [25] For web applications that have browsers as their target platform, this is difficult to implement and for our animation system this would be an overkill.
Nevertheless, we solve the problem with an approach based on blend trees.

We are introducing a state type, which contains a simplified state machine in the typical blend tree structure with new states in the form of Common States. The concept is shown in figure 7.

External Behaviour Externally, this Hierarchical Variation State behaves like a normal single state for the overall Action State Machine. An exemplary structure is shown in figure 7 A.
The internal behaviour of the state has no direct effect on the behaviour of the action state machine. However, it is responsible for the animation of the object.
The Hierarchical Variation State does not contain a fully-fledged sub-Action State Machine, which gets the control from the superordinate State Machine and is exited via an exit state (Compare Sub Action State Machines 9.2.3 on page 13).

The current Action State Machine performs outgoing transitions of the Hierarchical Variation State if they are fulfilled, regardless of the internal state.

Internal Structure Internally, the state tree can consist of several layers:

Base Layer The Base Layer is active the whole time the Hierarchical Variation State state is active.

Default Layer The Default Layer is active if no other state is active. For this reason, it has a default transition without conditions.

Further Layer There can be any number of additional layers defined by the developer.
These layers have in-transition with conditions, which define when they can become active.

The internal structure is shown in figure 7 B.

Internal Behaviour The layers basically behave according to these rules:

- A layer has a higher priority than all layers below it in the list.
- A layer becomes active when its in-transition is fulfilled.
- A layer is actually applied if it has the highest priority of all active layers.

The layers behave in a similar way to normal states in an Action State Machine. At least one state is active and which one is active is determined by Transitions.

But there are special cases: It can be useful to have several states active. For example, the shape of an object can be defined in the first layer state. The other layers deal with the user's interaction and indicate them only by adjusting the color of the object, e.g. hover effects and click animations. The shape should be always the same as defined in the first layer. Determining the shape of each layer separately is time-consuming and prone to errors. It makes sense to use a cascading principle: Basic attributes are always applied and

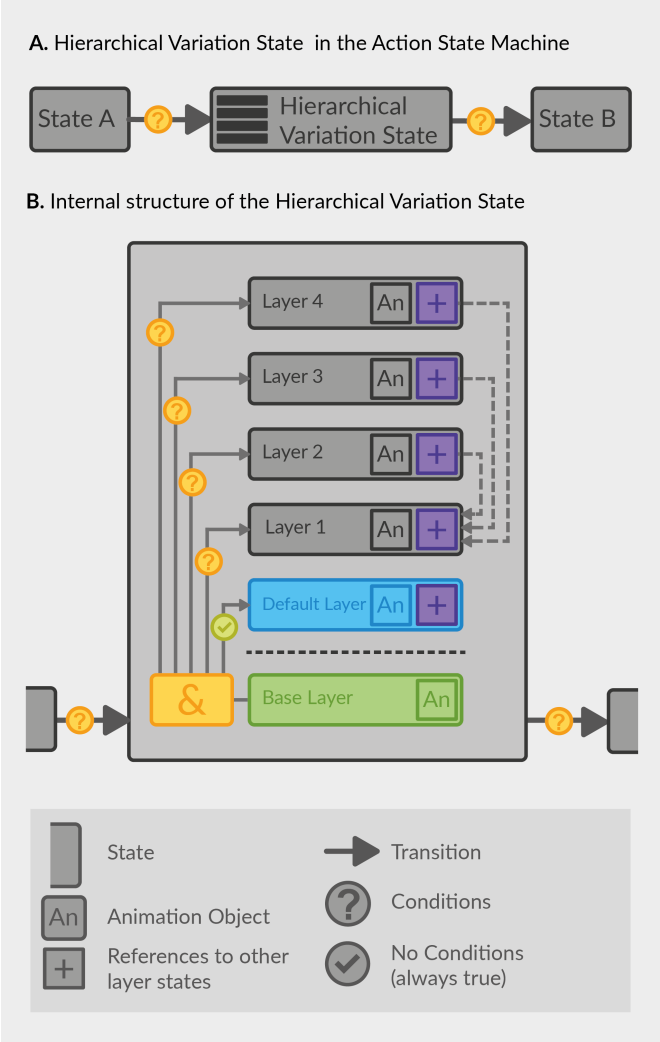

Fig. 7. The concept of Hierarchical Variation States.

higher layers can overwrite and supplement them.
Therefore, the following aspects are introduced:

- Besides the first layer, there is a base layer which is always applied. Regardless of which state is currently active (See figure 7 B).
- Each layer can reference different layers which are also executed when the original layer is active. Even if the selected layers themselves are not active. (See figure 7 B).

We provide two methods to ensure a uniform appearance for layer states without the need to implement cumbersome solutions with repetitions:

- The base layer is always applied and overwritten if necessary.
- Multiple layers can reference another layer, which is always activated with them.

Execution Order The base level is executed first. The rest of the execution is from back to front. So first the states referenced by the active state with the highest priority are executed, then the active state with the highest priority itself is executed.

This ensures that the properties of the most specialized layers can overwrite the more general layers if a value is set more than once.

9.2.3 Sub Action State Machine State

This state is constructed in the same way as Common States, except that an entire state machine serves as content instead of an animation object.

When this state becomes active, control is given to the Action State Machine referenced here. The sub-Action State Machine starts its execution from its entry state. The superordinate state machine is in an idle state until the sub Action State Machine reaches an exit state and returns control to the top level Action State Machine.

9.2.4 Transitional State

A Transitional State provides the simple possibility of animated transitions. It links two states with each other via an animation.
This approach allows complex transitions between two animations.

For the app developer, this construct is like a normal transition with extended possibilities. Thus, it should only be able to be added to existing transitions between two states in order to extend this transition by a Transitional State.
Internally, this construct is considered a predefined structure from a state as a node and two transitions as edges. An exemplary structure is shown in figure 8.
The state of the Transitional State is a Common State or Hierarchical Variation State which is non-looped and with an Exit Time of 1.0.
It has exactly one incoming transition with conditions defined in the Transition of the Transitional State and exactly one outgoing, unconditional Transition to the destination state.
Its transition duration is the duration of the state. Both the incoming and outgoing transition should have a transition duration of zero, as other values would result in an additional delay.

Generally seen a Transition with a Transitional State behaves similar to a normal transition: When a condition is fulfilled, the transition from a state to a target state takes place within a certain period of time. Only the internal structure differs from normal transitions.

9.2.5 Entry State

The Entry State is a special state. It has no content and is only connected to other states via transitions.

The Entry state is the entry point of the Action State Machine. It has exactly one transition to another state. This

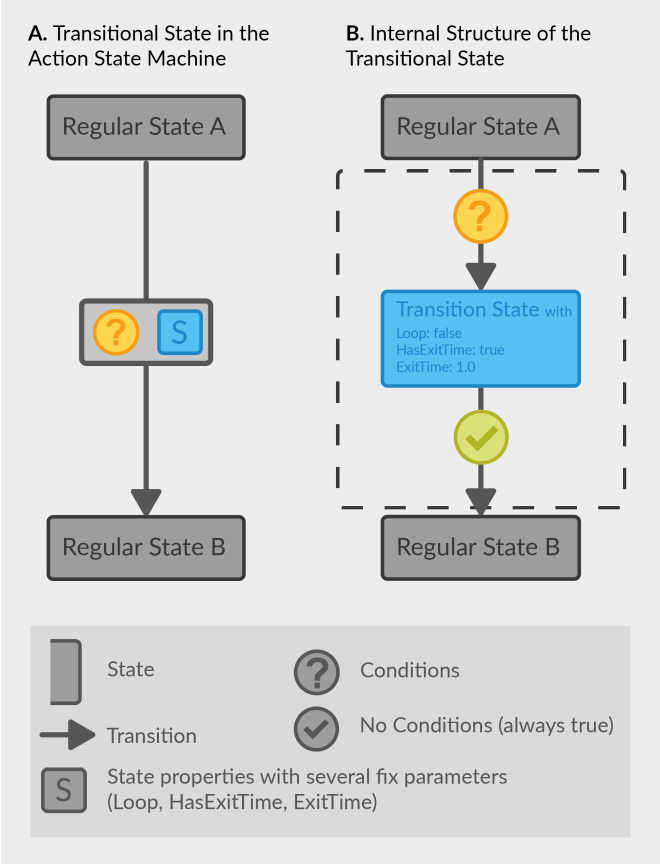

Fig. 8. The concept of Transitional States.

other state is the default state. At runtime, the Action State Machine immediately transitions to the default state which becomes the active state.
If no transition has been set, the first created state should be used as the default state.

9.2.6 Exit State

The Exit State is a special state. It has no content and is only connected to other states via transitions.

The Exit state defines the end point of an Action State Machine. The state has only incoming transitions.
In the case of nested state machines, control is returned to the higher state machine that has activated this state machine if the exit state becomes active. Otherwise, if the current State machine is not part of a nested state machine system, the execution is stopped for the time being and the state machine falls into an idle mode. No further animations are available for playback.

9.2.7 Any State

The Any State is a special state. It has no content and is only connected to other states via transitions.

The Any State represents, as the name suggests, any other state. It implicitly has only outgoing transitions to other states (You can not go from a state to ” any state ”). [26]

The state machine continuously checks for each active state whether an outgoing transition is true and performs it if necessary. The Any State extends the list of transitions of the active state by its own list of transitions. So in addition to the transitions of the active state, the Action State Machine also checks and executes all transitions of the Any State.
This can be advantageous if you want to go to a special state in a situation, no matter which state you are currently in. Otherwise you would have to insert a transition from all other states to the special state.

9.3 Transitions

Transitions provide one-to-one relations between two nodes (states and conditional hubs) of the Action State Machine. [26, 27] They are implemented as first order entities.

9.3.1 Transition Parameters

Transitions have the following properties:

Source State The source state of the transition
Destination State The destination state of the transition.
Is Bidirectional This property determines whether the transition can also be taken from the destination to the source.
Is Negated Bidirectional This property determines whether the transition can also be taken from the destination to the source with negated conditions.
Priority The priority of this transition. This property is relevant if multiple transitions are fulfilled.
Mute Define whether the Transition is muted or not. Muted transitions are ignored in all calculations.
Duration The total duration of the transition from invocation until the target state is the current state.
Delay The delay of the start of the transition process.
Has Exit Time This property specifies whether the exit time is relevant and should be checked before the conditions.
Exit Time A time value greater than zero which defines the temporal condition whether this transition can be taken or not.
Conditions A list of Condition Statements. The transition can be taken if all Condition Statement are fulfilled. The list can also be empty. If the List is empty, the condition of this transition is always true.

9.3.2 Condition Statements

A Transition can have any number of Condition Statements, including zero statements. If the Transition have no Condition Statement at all, the condition of this Transition is always fulfilled and only other aspects are considered such as the Exit Time.

A Transition Condition Statement can be any test against a State Property value of the Application State Model which outputs a binary value indicating a truth value.

A condition statement has the following structure:

- A State Property value of the Application State Model which is tested in the Condition Statement
- A conditional predicate that depends on the type of the State Property. For integer numbers, for example, the conditional predicates *larger than*, *smaller than*, or *equal as* are possible.
  A conditional predicate is not always necessary. A boolean variable can already form a valid condition statement.
- An optional parameter value. Some conditional predicates, for example, *greater* or *smaller than* require a value to resolve the statement. The parameter value has the same type as the State Property value.

9.3.3 Transition Direction

The normal flow of a transition goes from source state to destination state.

The bidirectional property can indicate that the transition is a bidirectional transition.
Internally, a copy of the state is then created interchanged with the source and destination state. All changes to the original transition are automatically applied to the copy. The copy is treated internally as a normal transition.

The negated bidirectional property can indicate that the transition is a bidirectional transition with negated conditions for the transition from destination to source.
Internally, a copy of the state is then created interchanged with the source and destination state and negated conditions. All changes to the original transition are automatically applied to the copy. The copy is treated internally as a normal transition.

9.3.4 Competing Transitions

A state can have several outgoing transitions. This has the consequence that the conditions of several outgoing transitions can be fulfilled at the same time. Since the state machine can only have one state at a time, only one transition can be performed. In such a situation, it is necessary to determine which transition will be performed.
Transitions have a priority property to solve this problem. Transitions with a higher priority are preferred to those with a lower priority if the conditions of both transitions are met.

A state should not have outgoing transitions with equal priority. All transitions should be brought into a clear hierarchy in which each priority occurs only once. A mechanism that checks whether the priorities of all transitions are consistent and outputs warnings if necessary is desirable.
In order to prevent unnecessary system failures, the system, nevertheless, takes the first transfer of the highest priority with fulfilled conditions to keep the state of the action state machine consistent and outputs an error message.

The priority approach makes it possible to optimize the system. Since the highest priority transition with fulfilled conditions is taken irrespective of the other results, no further transitions would be necessary.
Following this it makes sense to sort the transitions by priority in descending order and start with the highest priorities. As soon as the conditions of a transition are met, the tests can be aborted and that transition can be performed.

9.3.5 Exit Time

The exit time approach adds a precondition that must be met for the Transition in order to be considered for a state transition. If the exit time condition is not fulfilled, the conditions of the Transition are not checked at all.

The HasExitTime property determines whether the Exit Time property should be checked before evaluating the conditions. If this is true, the exit time conditions must be fulfilled for the system to check the conditions.

The Exit Time property defines the condition which must be fulfilled. The type of condition depends on the value:

- value ≤ 1: The number represents the progress of the animation in percent. For example, 0.75 stands for 75% progress of the animation.
  In addition, this condition is applied to each pass of the animation when the animation is played back in a loop. In our example, this means that the transition can only be taken if the animation playback is between 75% and 100% of the animation, regardless of how many times the animation has already been played.
- value > 1: The number represents the absolute playback time of the animation in seconds. For example, 2.5 stands for 2.5 seconds of animation progress. This time ignores the length of the clip and includes repetitions. For a clip with a length of 0.5 seconds, which is played in a loop, this means in our example that the transition can only be taken when the clip has been repeated 5 times.

This system, in conjunction with a property value of the state which determines whether or not to play a clip in a loop, covers all cases that affect an Exit Time. [26]

9.4 Conditional Hub

Conditional Hubs provide more advanced transition possibilities. They serve as a connection point with their own conditions between arbitrary states.
In addition to the one-to-one relationships of transitions, Conditional Hubs can model one-to-many, many-to-one and many-to-many relations. [27]

Conditional Hubs are the only elements besides States that are represented as nodes in the graph of the Action State Machine. It has no content and is connected to other states via transitions.

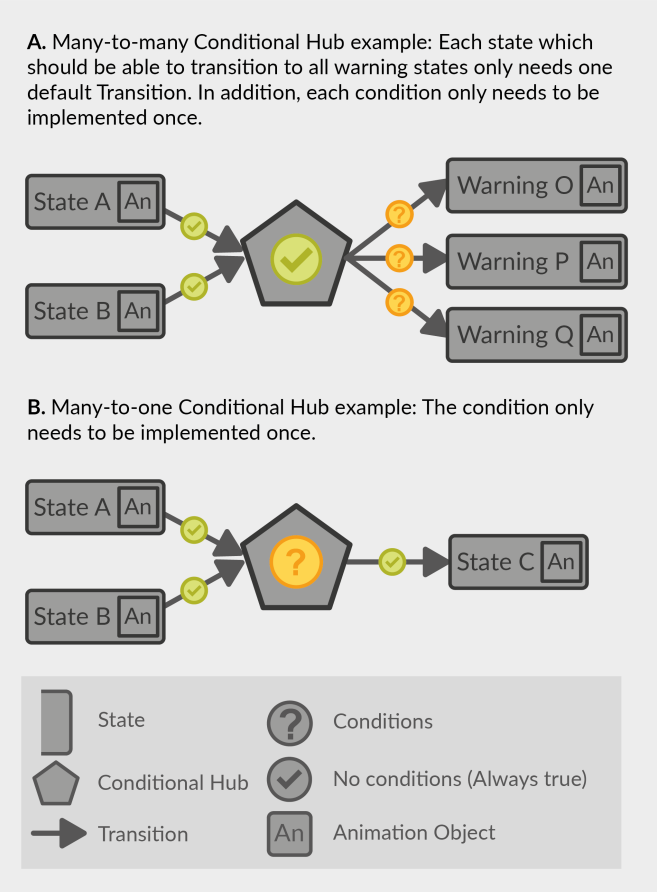

Fig. 9. A. Many-to-many Conditional Hub example. B. Many-to-one Conditional Hub example

9.4.1 Conditional Hub Conditions

A Conditional Hub itself can have Condition Statements equivalent to those of Transitions.
These Condition Statements must be fulfilled in addition to the own conditions of an incoming Transition. This applies to all incoming transitions.

9.4.2 Conditional Hub Use Cases

This behaviour can be used to the advantage in various situations:
Generally required conditions can be specified once within the Conditional Hubs and Transitions only have to implement special conditions which depend on their source state.
For one-to-many relations, it makes no difference whether the conditions for the incoming transition are implemented within the Conditional Hubs or whether they are implemented in the incoming transition itself.

Many-to-one relations with one general set of conditions can be easily implemented by only assigning conditions to the Conditional Hub in combination with always true transitions for all incoming and the outgoing transition.
It can be avoided that multiple transitions with the same conditions and the same target state have to be created unnecessarily. An example is shown in figure 9 B.

Many-to-many relations can be easily implemented by assigning conditions to the Conditional Hub in combination with unconditional incoming Transitions and outgoing transitions with various conditions. An example is shown in figure 9 A.

9.4.3 Conditional Hub Behaviour

A Conditional Hub represents a node, but must never be an active state itself, since it does not provide an animation. Therefore, in addition to the Transition conditions to the hub (conditions of the transition from state to hub and the conditions of the hub itself), at least one outgoing Transition must also be fulfilled from the hub to a fully-fledged state. Otherwise, it cannot be transitioned to the hub.

A default transition is conceivable. Each hub can mark an outgoing transition as default. This transition is always selected if no other outgoing transition is accomplished.

9.5 Runtime Behaviour

At runtime, the Action State Machine behaves according to its Application State Model and defined state machine graph.

From the Entry State, the first current state is determined immediately after start. From now on during runtime, the Action State Machine is always in an active state until it reaches an endpoint.

The Action State Machine always resolves its current state or perform a Transition.
In order to behave as expected, the Next State is introduced. It is needed to describe the internally state of the state machine and the behaviour that should be applied.
The runtime behaviour is defined by these two properties.
Basically, the system behaves in two steps in each iteration.

9.5.1 Determination Step

The system behaves depending on the Current State and Next State when starting an iteration. Two situations can occur:

- The Next State is not set: Currently no transition from the last iteration is active.
  It can be checked whether the state machine has to change to a new state or stay in the current. A Transition Test is performed.
  If there is a transition with fulfilled conditions, the Next State is set to this state.
  If no outgoing transitions are fulfilled, the Action State Machine should remain in its current state and the Next State remains unassigned.

- The Next State is already set to a state: Currently, a transition from the last iteration is not yet resolved. This state has to be resolved in order for the Next State to

become the Current State. In this situation, the already present Transition will be accomplished in the Action State. There are no further Transition Tests.

9.5.2 Action Step

The first step was to analyse the current situation. This step defines the further behaviour of the Action State machine in order to resolve the current state:

- The Next State is not set: The system will remain in its current state. The content of the Current State can be applied. The processing power is forwarded to resolve the animation content of the Current State.
- The Next State is set to a state: The system should transition to a new state depending on the information of the Transition between the Current State and the new state. The corresponding behaviour for resolving the situation is called up.

9.5.3 Transition Tests

A Transition Test should check all outgoing Transitions of the current state. This includes all outgoing Transitions of the Current State itself and the Any State.

A Transitional Test requires the following optimizations at the beginning of the runtime to work correctly. All relevant Transitions must be determined for each state. This list should then be sorted in descending order of priority. Muted Transitions are excluded from this list.

If any outgoing Transition has a Conditional Hub without outgoing default Transition as its destination, further Transitions of the Conditional Hub must be included in the test to ensure a consistent state of the Action State Machine. The Condition Statements of all outgoing Transitions can be linked together by logical *Or* operators and attached to the statements of the original Transition. This Transition is then only taken if a new state can be reached.

Basically, a Transition Test contains the following steps:

1. Get the sorted Transition list of the current state.
2. Start at the element with the highest priority and iterate through the list. When the end of the list is reached, no transition was fulfilled. The Transition Test does not change the current state properties and returns.
   (a) Check the Exit Time conditions
   (b) Check all Condition Statements.
   Cancel the check on the first statement that is not fulfilled and continue with the next Transition.
   (c) All checks were successful. The Transition is fulfilled. The list iteration can be exited and continued with the next point.
3. Set the Next State to the destination state of the Transition.

10 Description Modules

Description Modules are code fragments that are integrated into the application code. They describe the state of the application without intending specific animations or needing to know the current state of the system.
The Description Modules are based on the property based approach of the animation system (see 5.2 on page 7).

A module consist of individual statements, whereby each statement sets one State Property of an Animation Controller.
They specify the Animation Controller, the identifier of the State Property and the new value for the variable.

10.1 Code Structure

It is up to the application developer whether he integrates the assignments scattered throughout his code, integrates larger blocks with several assignments, or creates his own script that sets all the values.
The modules can thus adapt to the architecture of the application and correspond to the structure of the sources of animation intentions (see 3.2 on page 3).

10.2 Consistency

The set of all Description Modules should describe a consistent state of the application. Therefore, miscellaneous code sections should not provide contradictory descriptions. It makes sense to recognize situations when a value is set more than once in a frame or when a property is set with different, inconsistent values. In both cases there is probably some kind of error or inconsistency in the Description Modules.

10.3 Conflicting Animation Intentions

Conflicting events over animation intentions cannot occur because the program code only describes the state and does not directly call animations or status changes. The Animation Controllers always have exactly one consistent state. So the Action State Machines can always work properly.

11 Conclusions

We have provided a design specification for an animation system which matches the requirements of the usage of animations in web applications in a dynamic and interactive environment and common interactive interfaces. Our system is based on proven concepts of established animation systems from the unity game engine [26] and the unreal game engine [27] as well as common game engine animation system aspects as provided by Gregory [25].

12 Outlook

This paper has merely provided an initial design specification of the animation system.
The system can be improved. Many aspects can be precalculated. A tool with a graphical user interface should be

provided to facilitate the animation creation process.

Subsequently, the system should be extensively tested and evaluated with regard to normal software requirements such as performance and reliability.
In addition, the application in a real environment should be tested. Both the use of the system by the application developer and the user experience of the user of the end product should be considered.

References

[1] Kurlander, D., and Ling, D. T., 1995. Planning-based control of interface animation. Tech. rep., Microsoft Research, One Microsoft Way, Redmond, WA 98052, United States of America.

[2] Dragicevic, P., Bezerianos, A., Javed, W., Elmqvist, N., and Fekete, J.-D., 2011. "Temporal distortion for animated transitions". In *CHI '11 Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM New York, pp. 2009–2018.

[3] Bederson, B. B., and Boltman, A., 1999. "Does animation help users build mental maps of spatial information?". In *Proceedings of the IEEE Symposium on Information Visualization*. pp. 28–35.

[4] Heer, J., and Robertson, G., 2007. "Animated transitions in statistical data graphics". In *IEEE Transactions on Visualization and Computer Graphics*, Vol. 13. pp. 1240–1247.

[5] Robertson, G., Card, S., and Mackinlay, J., 1991. "Cone trees: Animated 3d visualizations of hierarchical information.". In *Proc. ACM CHI 1991*. pp. 189–194.

[6] Tversky, B., Bauer Morrison, J., and Betrancourt, M., 2002. "Animation: Can it faciliate?". In *Int. J. Human-Computer Studies*, Vol. 57. pp. 247–262.

[7] Gonzales, C., 1996. "Does animation in user interfaces improve decision making?". In *Proc. ACM CHI 1996*. pp. 27–34.

[8] Baudisch, P., Tan, D., Collomb, M., Robbins, D., Hinckley, K., Agrawala, M., Zhao, S., and Ramos, G., 2006. "Phosphor: Explaining transitions in the user interface using afterglow effects.". In *ACM UIST 2006*. pp. 169–178.

[9] Palmer, S. E., 1999. *Vision Science - Photons to Phenomenology*. MIT Press.

[10] Chang, B.-W., and Unger, D., 1993. "Animation: From cartoons to the user interface". In *UIST 1993*. pp. 45–55.

[11] Klein, C., and Bederson, B. B., 2005. "Benefits of animated scrolling". In *CHI 2005 Extended Abstracts*. pp. 1965–1968.

[12] Oksama, L., and Hyn, J., 2004. "Is multiple object tracking carried out automatically by an early vision mechanism independent of higher-order cognition? an individual difference approach". In *Visual Cognition*, Vol. 11. pp. 631–671.

[13] Oksama, L., and Hyn, J., 2005. "Tracking multiple targets with multifocal attention". In *TRENDS in Cognitive Science*, Vol. 9. pp. 249–354.

[14] Sundar, S. S., and Kalyanaraman, S., 2004. "Arousal, memory, and impression-formation effects of animation speed in web advertising". In *Journal of Advertising*, Vol. 33. pp. 7–17.

[15] Vodislav, D., 1997. "A visual programming model for user interface animation". In *Visual Languages, 1997. Proceedings. 1997 IEEE Symposium on*.

[16] Robertson, G. G., Card, S. K., and Machinlay, J. D., 1993. "Information visualization using 3d interactive animation". In *Communications of the ACM*. pp. 57–71.

[17] Harrison, C., Zhiquan, Y., and Hudson, S. E., 2010. "Faster progress bars: Manipulating perceived duration with visual augmentations". In *CHI 2010*.

[18] Myers, B. A., 1985. "The importance of percent-done progress indicators for computer-human interfaces". In *Proc. ACM CHI'85 Conf.* pp. 11–17.

[19] Nielsen, J., 1994. *Usability Engineering*. Academic Press.

[20] Tversky, B., Bauer Morrison, J., and Betrancourt, M., 2002. "Animation: can it faciliate?". In *Int. J. Human-Computer Studies 57*. pp. 247–262.

[21] Thomas, F., and Johnston, O., 1981. *Disney animation: The Illusion of Life*. Abbeville Press.

[22] Baecker, R., and Small, I., 1990. "Animation at the interface". In *The Art of Human-Computer Interface Design*. Addison-Wesley, pp. 251–267.

[23] Lang, A., 2000. "The limited capacity model of mediated message processing". In *Journal of Communication*. pp. 46–67.

[24] W3c standards. `https://www.w3.org/standards/`. Accessed: 2017-08-27.

[25] Gregory, J., 2009. *Game Engine Architecture*. A K Peters, Ltd, 5 Commonwealth Road, Suite 2C, Natick, Massachusetts, United States of America.

[26] Unity3d documentation. `https://docs.unity3d.com/Manual/index.html`. Accessed: 2017-08-25.

[27] Unreal engine 4 documentation. `https://docs.unrealengine.com/latest/INT/`. Accessed: 2017-08-25.

[28] Lewis, P., and Thorogood, S. Animations and performance. `https://developers.google.com/web/fundamentals/design-and-ux/animations/animations-and-performance`. Accessed: 2017-09-27.