ANIMATION OUT-OF-THE-BOX: AN APPROACH FOR SPATIO-TEMPORAL ANIMATION BASED ON A DECLARATIVE LANGUAGE

GUSTAVO ADOLFO GARCIA CHAPETON
March, 2014

SUPERVISORS:

Dr. Ir. Rolf A. de By
Dr. Barend J. Köbben

ANIMATION OUT-OF-THE-BOX: AN APPROACH FOR SPATIO-TEMPORAL ANIMATION BASED ON A DECLARATIVE LANGUAGE

GUSTAVO ADOLFO GARCIA CHAPETON
Enschede, The Netherlands, March, 2014

Thesis submitted to the Faculty of Geo-information Science and Earth Observation of the University of Twente in partial fulfilment of the requirements for the degree of Master of Science in Geo-information Science and Earth Observation.
Specialization: Geoinformatics

SUPERVISORS:

Dr. Ir. Rolf A. de By
Dr. Barend J. Köbben

THESIS ASSESSMENT BOARD:

Prof. Menno-Jan Kraak (chair)
Dr. A.G. Toxopeus

ABSTRACT

Cartographic animation have been pointed out by several authors as an ideally suited approach for the study of dynamic phenomena. In recent years, different systems for animated web map production have been developed. But we identified some deficiencies with them: In general, the procedure to create the animated maps is time-consuming, hence it is not appropriate when a large number of maps is required; most of the systems require the data to be in a specific data format, so they are not capable to consume data directly from regular data stores; and performance issues when working with large datasets have been pointed out. To overcome the identified deficiencies, we propose an approach for animated web map production based on a declarative language. We designed the Animation SPECification Language (ASPEC-L) and a toolset capable to produce animations from specifications written on it. The development of a prototype of the system, demonstrates that this approach can be implemented in a functional system. The prototype was used to build animations for three application cases: Lilac blooming in USA, Swainson's hawk migration over the Americas and Hurricane movement on the Pacific Ocean. The usage of the prototype in these application cases demonstrated that our approach is broadly applicable, and that it provides a systematic and consistent procedure for animation production that has the potential to overcome the identified deficiencies.

Keywords

dynamic phenomena, visualization, spatio-temporal animation, cartographic animation, animation production, declarative language

ACKNOWLEDGEMENTS

Special thanks to my supervisors Dr. Rolf A. de By and Dr. Barend Köbben, for their dedication, support and guidance during this project.

Thanks to the NICHE Project GTM042 “Strengthening of Human Resources in Land Administration (LA) in Guatemala”, for sponsoring my studies at the Faculty of Geo-information Science and Earth Observation (ITC), University of Twente.

Thanks to the institutions that provided the data for this project: World Data Center for Paleoclimatology, Boulder and NOAA Paleoclimatology Program; MOVEBANK project of the Max Planck Institute for Ornithology; and UNISYS.

Special thanks for all the people who made my time at ITC a special and unforgettable experience: Sana, Siddhi, Zahra, Leila, Neda, Eva, Eduardo, Manuel and Andres.

Finally, my greatest thanks to my family in Guatemala. My parents, Ramiro and Maria, and my brothers Carlos and Cyndi. For encourage me to go abroad to continue with my studies and for all their advices and support during this period.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1
Introduction

1.1 BACKGROUND AND PROBLEM STATEMENT

Nowadays, spatiotemporal (ST) datasets are being produced for a large variety of phenomena, either in raster or vector format. This is due to the advances in technology that make it possible to capture, store and share this kind of data easily. The availability of these datasets gives us the chance to gain insight in the dynamics and patterns of the phenomena in study, but for this aim, we should apply the appropriate approaches to make use of the data.

In general, datasets can be exploited making use of different approaches (e.g. statistical analysis, static and dynamic visualization, and animation). But in particular when dealing with ST datasets, animation provides an intuitive way to represent dynamic phenomena (Harrower & Fabrikant, 2008), which makes it ideally suited to visualize and analyze the data (Sayar, 2012). Nevertheless, there are several considerations for building useful animated maps (Harrower & Fabrikant, 2008), among them the size of display, type and number of objects presented, type of animation, frame rate (Dong et al., 2012), the level of control provided to the user (Cinnamon et al., 2009) and the platform on which the output is generated (De Ambros et al., 2011).

It makes sense to think about animated web maps. First, because as Sayar (2012) stated, animation is ideally suited to represent dynamic phenomena; and second, the web makes possible to reach a broader community of users as indicated by De Ambros et al. (2011). Furthermore, we know it can be done, because it has been done by authors such as Becker (2009) and Sayar (2012), they designed and developed tools for animated web map production. Additional examples of such tools are WMS animator (GeoServer, 2013), Animaps (Animaps, 2011) and i2maps (i2maps, 2010).

Different systems for animated web map production have been developed in recent years. But we identified some deficiencies with them:

1. In general, the procedure to create the animated maps is time-consuming, hence it is not appropriate when a large number of animations is required.

2. Most of the tools require the data to be in a specific data format, so they are not capable to consume data directly from regular data stores.[1]

3. Performance issues when working with large datasets have been pointed out.

Given the importance of animation for visualization and analysis of ST datasets, we must conduct research into designing tools that overcome the identified deficiencies.

[1] OGC Web Services (OWS), shapefiles, spatial databases, etc.

1.2 A SOLUTION BASED ON A HIGH-LEVEL SPECIFICATION LANGUAGE

To overcome the identified deficiencies, we propose an approach to produce animated web maps based on a declarative language. The Animation SPECification Language or ASPEC-L, can be described as a declarative domain-specific language to specify web animated maps inspired by MapServer Mapfile syntax. An example of an animation specification is shown in Listing 1.1.

Listing 1.1: ASPEC-L example code.

```
ANIMATION
    START_TIME "2010-03-15T14:50:58Z"
    END_TIME "2010-04-15T14:50:58Z"
    DURATION "00:00:15"
    FORMAT "CZML"
    BBOX 20 -50 55 -130
    BBOX_BEHAVIOUR "AUTO ADJUST"
    ...

    LAYER
        TYPE RASTER
        DATASOURCE "WMS"
        OWS_URL "HTTP://www.someowsservice.nl/"
        DATA "Temperature"
        ...
    END

    LAYER
        TYPE "POINT"
        DATASOURCE "WFS"
        OWS_URL "HTTP://www.someowsservice.nl/"
        DATA "birds2010"
        ANIMATION_TYPE "INTERPOLATED"
        FIELD_GROUP_BY "bird_id"
        FIELD_TIME "time"
        CLASS
            SIZE 5
            COLOR 0 0 255 100
            ...
        END
        ...
    END
    ...
END
```

A mechanism to produce the animations based on the high-level description is needed. Such a mechanism is provided by a toolset designed specifically for this aim. The toolset is composed of three components: a text parser to analyze the high-level language, an animation code generator to produce the animations and a rendering component to play the animation on the output platform (see Figure 3.4).

As proof of concept a prototype of the toolset was built. To demonstrate the capabilities of the prototype, three application cases were selected:

1. Lilac blooming in USA.

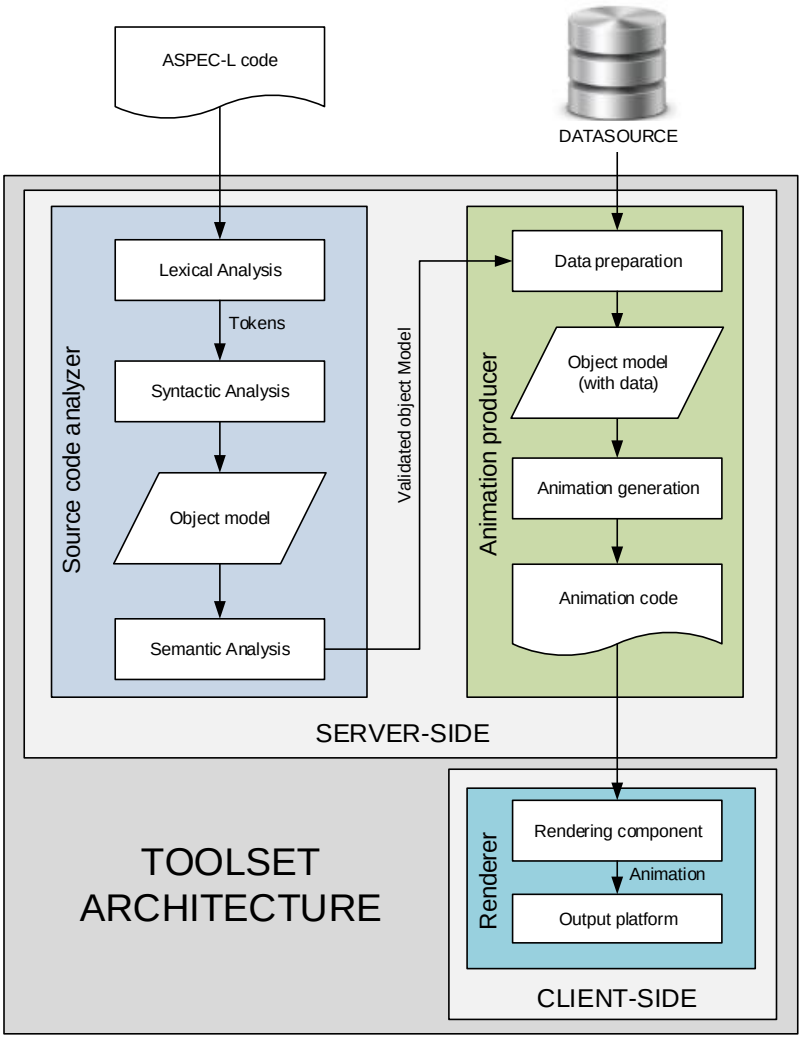


Figure 1.1: Toolset architecture.

2. Swainson's hawk migration in the Americas.

3. Hurricane movement on the Pacific Ocean.

A relevant aspect to point out about the application cases, is that they were chosen having in mind that they all use point vector data and require different kinds of animation. In the first case, the point features need to appear in fixed position, it is an animation that represents an existential change. The second case requires an animation for location change, it means points moving in space over time; this is a more complex animation that requires an interpolation algorithm to compute the intermediate positions between the ones stored in the ST dataset. And the third case, just as in the second case, an interpolated movement is needed, but additionally re-expression is required to depict the change of intensity. By re-expression, we mean the change of visual representation on screen.

1.3 RESEARCH OBJECTIVES

The general objective of the present work was to design an approach to produce animated web maps following a systematic and consistent procedure based on a high-level specification language, capable to consume data from regular data stores.

This general objective can be split down into the following specific ones:

1. To design a specification language for animated web maps.
2. To design a toolset for building animated web maps based on specifications on the above language.
3. To build a prototype of the toolset that implements the functionality to generate animated web maps, as specified, from datasets in vector format containing point features and from raster datasets.
4. To use the prototype in the proposed application cases to evaluate whether the implemented features work according to the design specifications and to test the performance of the toolset.

1.4 RESEARCH QUESTIONS

Related to objective 1:

1. Which features of the animations should be taken into account to design the language?
2. Which lexical and syntactical elements should be defined to design a specification language for animated web maps?
3. Which design criteria can be applied to ensure the design of an extensible language?

Related to objective 2:

1. Which processes should be part of the toolset?
2. What is the minimal information required to build each type of animation?
3. Which design criteria can be applied to ensure the creation of an extensible toolset?

Related to objective 3:

1. How will the algorithms for different types of animations be integrated in the toolset?

Related to objective 4:

1. How suitable are the produced animations to study the phenomena represented?
2. How do different settings for a map affect the performance of the toolset?
3. How well does the toolset perform with increasing dataset sizes?

1.5 INNOVATION

The innovation in the present research project is the design of a high-level language for map animation specification, and based on it, the design and realization of a toolset for animated web map production.

1.6 METHOD ADOPTED

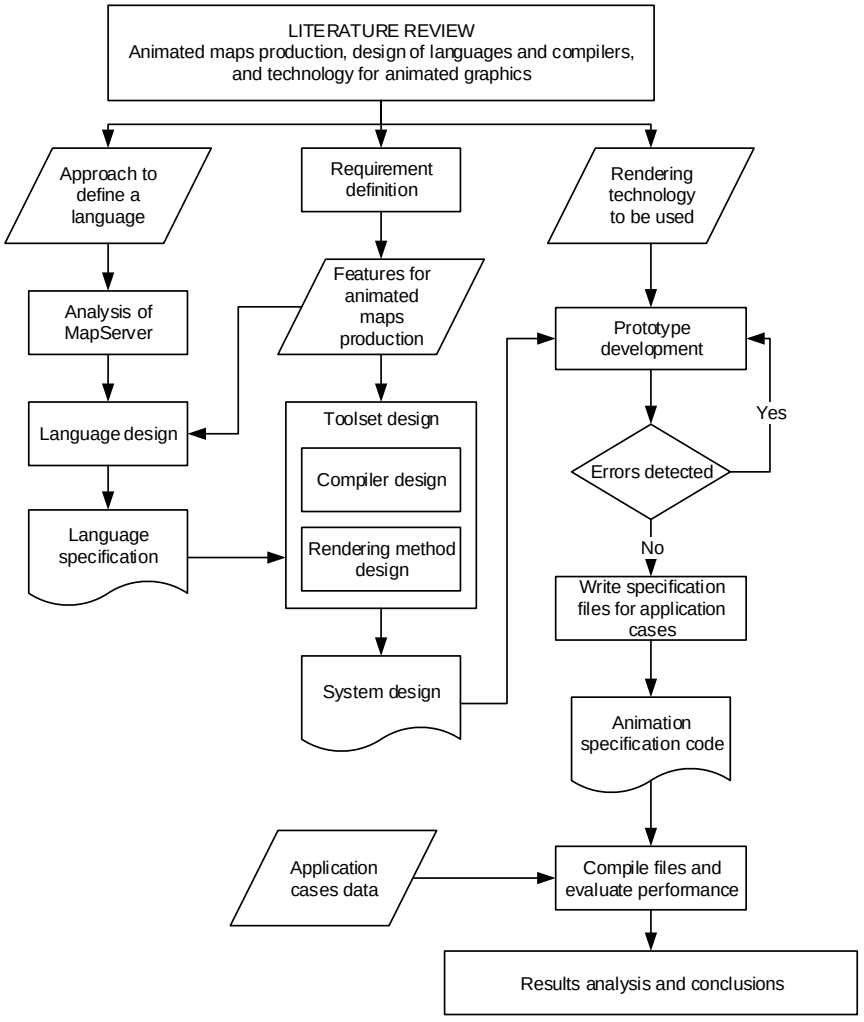


Figure 1.2: Research workflow.

To conduct the present research, the following activities were executed (see Figure 1.2):

1. Literature review: Literature on animated maps production, design of languages and compilers, and technology for animated graphics were reviewed. From this activity the following outputs were produced: selection of an approach to design the language, characteristics to be included when modeling the toolset and selection of rendering technology for the animated web maps.

2. Requirements definition: The characteristics of the system were described.

3. Analysis of MapServer: As mentioned before the designed language is inspired by the MapSever Mapfile language, because of this reason the structure of such files, as well as the software used in MapServer were studied.

4. Language design: based on the outputs produced in the previous activities, the language was designed, which means that the lexical and syntactical elements of the language were defined.

5. Toolset design: Based on the designed language, a toolset to produce animations was designed.
6. Prototype development: Based on the design of the toolset, a prototype was built. In this step, an incremental approach was applied to build the prototype.
7. Prototype debugging: The prototype was tested to find and fix bugs.
8. Applying prototype to application cases: Animations for the application cases were built using the prototype.
9. Performance evaluation: The toolkit was tested by producing animated web maps with different settings and dataset sizes.

The phases described above present the logical sequence of the project, but not necessarily a chronological sequence, because an evolutionary approach was used in developing the project.

1.7 THESIS OUTLINE

Including this introduction, the present thesis is composed of six chapters. In **Chapter 2**, we present the principles and desired characteristics for animated maps. That chapter serves two purposes, first to define the general context for animated maps and second to provide a starting point for the design stage. In **Chapter 3**, we propose an approach for animated map production based on a declarative language. This chapter includes the design of the Animation SPECification Language (ASPEC-L) and the design of a toolset to produce animations from specification written in ASPEC-L. In **Chapter 4**, as proof of concept, we describe the development of a prototype that implements our approach for animated web maps production. In **Chapter 5**, the results of using the prototype in three application cases are presented. Finally, in **Chapter 6**, conclusion and recommendation for further research are discussed.

Chapter 2

Spatio-temporal animation: principles and characteristics

2.1 DYNAMIC GEOGRAPHIC PHENOMENA

The changes that take place as part of a phenomenon, can be used to characterize its dynamics. In the context of this research project, what we mean by change is a variation of one or more of the characteristics of the phenomena under study, where the characteristics are geometric, thematic and temporal (Blok, 2000). A simple example can be a car on a trip, it is moving along a path (change in geometric characteristic), while it happens, it is consuming its fuel (change in the thematic characteristic of fuel level) and it occurs in a time span.

From the example above, we can point out time as the key concept of studying dynamic phenomena. The concept of time and space, and the relation between them are not new, they can be traced back to ancient times. The discussion about time and space, has considered them as absolute and relative concepts; as real objects in themselves or as mere product of human imagination; structured as an infinite continuum or as a finite past with a beginning; as a linear or cyclical succession of events, or even as a branching succession of events; and this discussion continue in the present days, and the question still remains "What is time?". Time seems to be a familiar concept for anyone, however, as indicated by Kraak and Ormeling (2011) it is not a straightforward concept to be defined. The Oxford dictionary defines time as: "the indefinite continued progress of existence and events in the past, present, and future regarded as a whole." Depending on the phenomena of study, it is useful to conceptualize time in different ways, and because of this reason, different kinds of time have been proposed by authors. There is no agreement about the basic types of time; however, linear, cyclic and branching are the most common types of time mentioned in the literature. Linear time corresponds to our natural perception of time, which is a continuous sequence of instants extending from the past to the future (G. Andrienko et al., 2010); cyclic time is conceptualized by the idea of repetition, examples of time cycles are days, weeks, years and seasons (Harrower & Fabrikant, 2008); and branching time, gives the possibility of represent and compare multiple scenarios from a phenomena (G. Andrienko et al., 2010). Additionally, the term time, can refer to 'world time', the moment at which the phenomenon happens in reality; 'Database time', the moment in which it is recorded in a database; or 'display time', the moment at which it is displayed in a visualization (Kraak & Ormeling, 2011).

Following with the example of the car, changes occur in two components of the phenomenon: spatial and thematic. Therefore, just as with time, change can be categorized. Work describing different types of change was done by Blok (2000), and based on it N. Andrienko et al. (2003) proposed the following classification:

1. *Existential change*: Refers to the events in which a phenomenon appears or disappears. For example, the car's creation in the factory and the car's demolishing in the car shredder, it

start to exist or appears at the moment it occurs.

2. *Spatial changes*: In this category, location and shape changes are included. They are changes that affect the geometric characteristics of the phenomena under study. For example, in the case of a wildfire, the burnt area grows as the fire advances, it is the shape of the phenomenon (burnt area) that is changing.

3. *Attribute changes*: These are the changes that affect the thematic properties of the phenomenon. For example, the car in the trip has different levels of fuel on each moment of the trip.

2.1.1 Visualizing dynamic geographic phenomena

Visualization is the translation from a dataset to a graphic representation of it, the term could refer to two different, but related activities, visual thinking and visual communication (Dibiase et al., 1992). The first can also be called data exploration and aims to provide insight in the dataset being visualized for this aim the user requires to have tools to interact and manipulate the visualization. The second is intended to visually communicate a message to a general and broader public, and in general no interaction tools are required.

Cartographers have developed several approaches to visualize dynamic phenomena. Among them, the most common in literature are static single maps, multiple maps, space time cube, and animated maps. On this research project we are interested specifically in animated maps, and in this regard Harrower and Fabrikant (2008), and later Sayar (2012) pointed out animation as ideally-suited to represent dynamic phenomena.

Nowadays, an increasing use of cartographic animation has led to the development of different techniques and types of animated maps. There is no single type of animation that fits all the purposes, hence, it is important to select the proper type of animation by the characteristics of the dataset and the aim of the visualization (Lobben, 2003). Additionally, it is important to make proper use of graphic and dynamic variables in the production of the animations. A brief description of these variables is provided in the following sections.

Graphic variables

Map production is a creative process, in which the cartographer needs to appropriately choose how to represent the data. Uncountable variations can be applied to the map symbology to communicate properly an intended message, these variations were classified initially by Bertin in 1967 and later extended by various researchers (Dibiase et al., 1992). They are known as graphic variables; the use of one or another depends on the type of data to be represented (see Figure 2.1).

Dynamic visualization variables

When representing a dynamic phenomenon by means of animation, traditional graphic variables are still applicable. However, in addition to them, Dibiase et al. (1992) and MacEachren (1994) defined duration, order, rate of change, frequency, display time and synchronization, the so-called dynamic visualization variables. They allow to represent specific aspects of dynamic phenomena that cannot be represented by traditional graphic variables. In this regard, Blok (2005) pointed out duration, frequency, order and moment of display as the most important ones, and considered rate

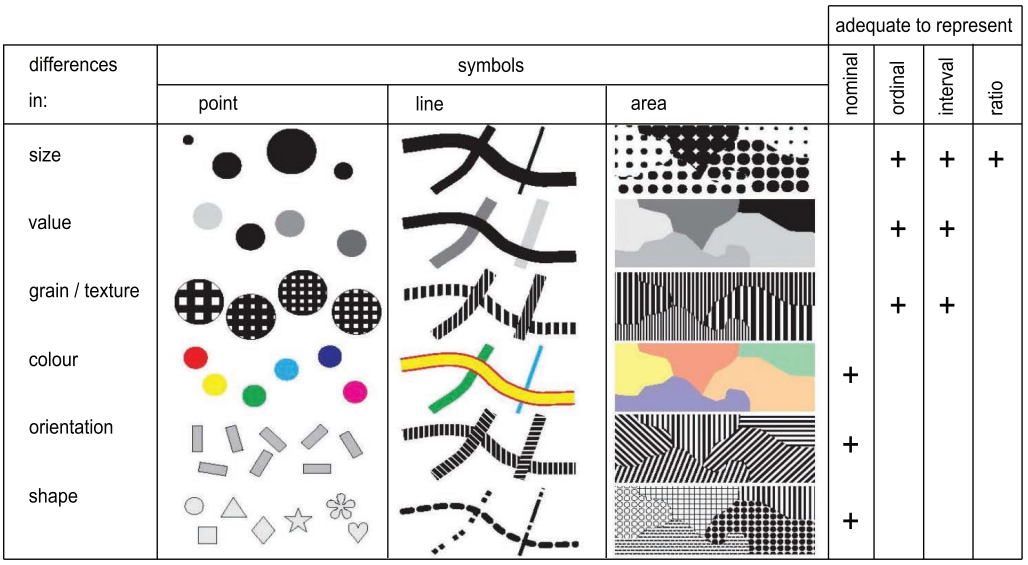


Figure 2.1: Graphic variables. From Kraak and Ormeling (2011).

of change and synchronization as consequences of these four. The dynamic visualization variables are defined as:

1. *Duration:* Number of units of animation time that a scene is in display. A scene is a static image representing the phenomenon in a specific instant of time (real-world time). By changing the duration of the scenes, the pace of the animation is changed.

2. *Frequency:* Repetition of identical states or changes in the animation per unit of display time.

3. *Order:* Structured sequence of states or changes in display time, which is not necessarily chronological. (Dibiase et al., 1992) stated that "The logic of chronological sequencing of scenes associated with a time-series data set is obvious; however, there are instances when ordering series by a metric other than chronology is logical and potentially fruitful in geographic analysis."

4. *Moment of display:* Position of a state or a change in display time.

Becker (2009) indicates that in temporal animation, most of the dynamic visualization variables are fixed as the animation simply represent the data and its temporal steps, and that they are more useful in the design of non-temporal animation. Additionally, he mention that they can be useful for emphasize change and attract user's attention.

2.2 SPATIO-TEMPORAL ANIMATION

2.2.1 Spatio-temporal data

Spatio-temporal data is the basis for spatio-temporal animation. The characteristics of spatio-temporal data can be described by using the conceptual framework proposed by Peuquet (1994).

This framework includes three dimensions to describe the data: thematic, spatial and temporal. These dimensions are directly related to the elementary questions What?, Where? and When? Figure 2.2 offers an example of how spatio-temporal data can describe a phenomenon and locate it both in space and time. Therefore, if we have data of the same phenomena at different times (a spatio-temporal dataset), we can build an animation to visualize it.

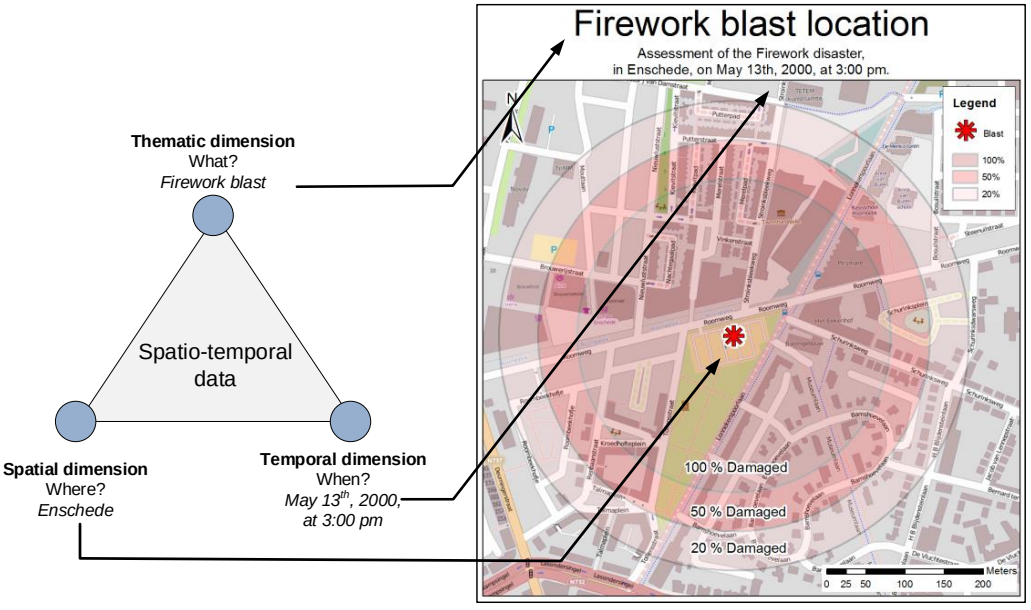


Figure 2.2: The dimensions of spatio-temporal data.

2.2.2 Definition and characteristics

In the geosciences, authors have referred to animation as cartographic animation, map animation, dynamic mapping, four-dimensional cartography, spatio-temporal displays, among others (Weber, 1991). As different terms have been used, different definitions have been written, Monmonier (1990) indicates that an animation is a "temporally-ordered sequence of views so that the map becomes a scale model in both space and time." Lobben (2003) states that the simplest definition for animation refers to "any moving presentation that shows change over time, space, and/or attribute," and Harrower and Fabrikant (2008) define animation as the "sequence of static graphic depictions that when showing in rapid succession, begins to move in a fluid motion." Despite that all these definitions are different in content, all of them implicitly or explicitly refer to the action of mapping change.

Animation as any other visualization approach is an intentionally scaled-down representation of the world (Harrower & Fabrikant, 2008). This statement is true for both the spatial and temporal dimension. In the spatial dimension the concept of scale is well-known and it is the ratio between the real-world lengths or areas and their equivalent in the visual representation, and in the temporal dimension it is the ratio between real-world time and animation time.

Animations can range from movie-like visualizations to highly interactive visualization systems (Harrower & Fabrikant, 2008). Depending on whether the aim of an animation is presen-

tation or exploration, different mechanisms for interaction can be provided. For a description of the most common exploratory functions in animated maps, see Section 2.2.5.

The amount of information that can be presented with animation is virtually unlimited. As opposed to static mapping, in which all the information is presented at once, in animated maps the information is displayed with the flow of the animation. Nevertheless, due to short-term visual memory constraints, it is important not to overload the user with information (Harrower & Fabrikant, 2008).

2.2.3 Advantages and disadvantages of animated maps

In contrast to other visualization techniques that are capable of only showing the macro-steps of the phenomena under study, animation is capable to show the micro-steps as well. This means that animation actually shows the process of the phenomena. According to Blok (2005), by using animation"one can actually 'see' how patterns shrink or expand, break up, the direction and speed of change, frequencies of events, etc." In the same sense, Ogao and Kraak (2002) commented that animations "play an intuitive role when used to view geospatial transitions as they happen in time as opposed to simply viewing the end states."

Many authors state that animated maps facilitate the identification and analysis of patterns. Harrower and Fabrikant (2008) indicate that animations are useful to identify patterns that are not evident in static representations, even for experts who are familiar with the dataset. Additionally, animated maps are useful in the identification of behavior that doesn't fit with the identified patterns. In this sense Ogao et al. (2002) stated that user attention is attracted to outliers in animated representations, and Lobben (2003) indicated that animation can be used to identify variations in a pattern that are not easy to detect from viewing the dataset as static maps or data tables.

As any other visualization technique, animation presents disadvantages as well. Among them are: Change blindness, which refers to changes that go unnoticed by the user. This can happen because of the view being interrupted (e.g., during eye movement) or the variations in the visual field are too weak or too slow to be noticed (Blok, 2000); long lasting animations can produce information overload, because as length increases, so does the difficulty to remember previous frames. Therefore, the user will not be able to analyze what is shown in the animation (Harrower & Fabrikant, 2008); the problem of split attention arises when the user needs to focus on two different elements of the animation, as for example the map and a temporal legend, which could lead to a misunderstanding of the animation content (Harrower & Fabrikant, 2008).

2.2.4 Taxonomy of animated maps

There is no agreement among authors about a single taxonomy for animated maps. In this section, we briefly describe four classification methods, which are: based on data type, based on time, based on level of control, and based on distinctive characteristics.

Classification based on data type

Becker et al. (2009) implicitly classify animated maps based on the type of data used to build the animation, the described types of animations are:

1. *Raster-based animation:* This type of animation is built by creating images from the raster data and displaying them as a series of static images one after another, creating in this way the effect of animation.

2. *Vector-based animation:* Based on vector data, the frames are created on-the-fly to produce the animation, while it is being played. Depending on the type of change to be depicted, it could be required to use interpolation to smoothen the transition between states in the animation.

Classification based on time

Harrower and Fabrikant (2008), and Kraak and Ormeling (2011) indicate that cartographic animation can be subdivided into temporal and non-temporal animation.

1. *Temporal animation:* It shows events in chronological order. In this animation type, the real-world time is scaled to animation time (temporal scale), therefore the phenomena are presented faster or slower than they occur in reality. An example in this category is the spreading process of diseases.

2. *Non-temporal animation:* Display time is not linked to real-world time, the dynamics of the animation is used to show geometrical or attributive characteristics of the phenomena in study, or spatial relationships on it. For example the fly-bys or fly-throughs animation to show relief characteristics of a study area.

Classification based on level of interactivity

Lobben (2003) and Harrower and Fabrikant (2008) stated that animation can be classified based on the level of interactivity.

1. *Animation for presentation:* The user is provided with little or no control over the progress of the animation. The user is usually provided with controls to play, pause and change the speed of the animation. We can call these movie-like animations.

2. *Interactive animation:* The user is provided with several options to control the flow of the animation, as before the user can play, pause and change the speed of the animation. Additionally, controls for manipulating the spatial extension (pan, zoom and rotate), navigate on the temporal dimension (temporal legends and go to time), play backward and forward, querying the data from the animation, among others may be included.

Classification based on distinctive characteristics

Lobben (2003) proposed a classification based on three criteria: time, variable[1] and space.[2] The dynamic or static quality of each criterion provides the classification basis (see Figure 2.3). The dynamic or static quality, can be explained as follows: in the case of time, if all the observations belong to a single moment in time, it is considered that time is static, otherwise it is dynamic; for variable and space, they are considered dynamic if their quality or quantity changes over time, otherwise they are static.

[1] Attribute dimension of data.
[2] Spatial dimension of data.

Animation Method	Characteristics					
	Time		Variable		Space	
	Dynamic	Static	Dynamic	Static	Dynamic	Static
Time-Series	X			X		X
Areal		X		X	X	
Thematic	X	X	X			X
Process	X		X		X	

Figure 2.3: Classification basis for distinctive characteristics classification. From Lobben (2003).

1. *Time-series animation:* Its most important characteristic is the depiction of change over time. In this type of animation the spatial extent of the study area is fixed, the symbolization of features doesn't change, but time does, giving the sensation of change by features that appear or disappear.

2. *Areal animation:* Contrary to time-series, the viewpoint is dynamic and time remains fixed. This meas that the animation shows a snapshot in time, which implies that symbolization is fixed, with the aim of highlight the characteristics of the phenomena at that specific moment in time. Example: Fly-bys over an area.

3. *Thematic animation:* Within a fixed spatial extent, this type of animation relies on the change of symbology to depict dynamics. If time is static, this animation type can be used to show for example different methods of classification or aggregation. In the case of dynamic time, the animation shows the evolution of a phenomenon over time.

4. *Process animation:* Time, variable and space may be dynamic. This type of animation aims to depict the process of development of the phenomena under study. Symbolization changes as effect of changes in the phenomena over time, and the spatial extent can change to show the whole phenomenon as it expands spatially or to focus user attention on a particular part of it.

2.2.5 Exploratory functionality on animated maps

When compared to paper maps, computer-based visualization tools present two new properties: dynamics and interactivity (N. Andrienko et al., 2003). Dynamics refers to the capability of the visualization to change its content based on some criteria. For example, a weather map can change every time it is visualized, to show the latest data available, in other words, the current weather. This property makes animation possible, because on-screen content changes based on display time. Interactivity is the capability of the visualization to react to user actions. For example highlight features on screen in response to a mouse click.

Cinnamon et al. (2009) performed a usability test, and the results point out that an animation can be more useful if control over it is provided. In this section, we briefly describe the most common types of interactive control included in animated maps (see Figure 2.4).

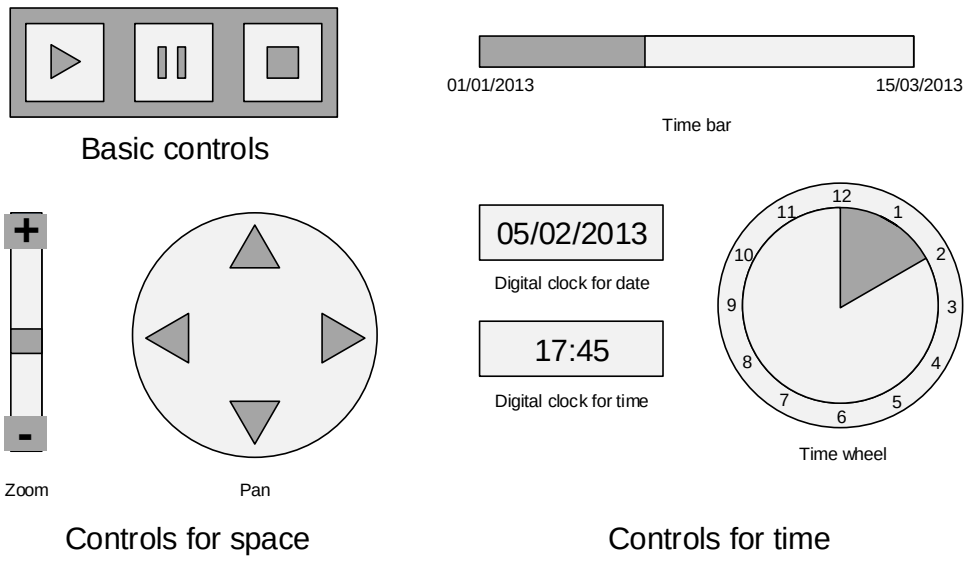


Figure 2.4: Most common controls in animated maps.

Basic control

The most common and basic controls to interact with the animation are play, pause and stop. Their functions are just the same as those in any media player. They are sometimes referred to as VCR-type control.

Spatial control

When studying a geographic phenomenon, the user could be interested to analyze either the whole area covered by the phenomenon or a specific spatial extent (Becker et al., 2009). To fulfill this necessity, controls to perform panning and zooming should be included.

Time control

Several mechanisms to control the time on animated maps are described in the literature. Harrower and Fabrikant (2008) describe temporal legends, which are controls that can be used to show the passage of time. They can be presented in the form of digital clock, time wheel or time bar. The advantage of a visual temporal legend over a digital clock is that it can provide in a glance information about the current time and where it is located in relation to the whole animation. Additionally, N. Andrienko et al. (2003) describe a control to specify the direction (forward and backward) in which the animation is played, controls to change the temporal extension, which can work as a temporal filter, and the option go to time, which provides a mechanism to 'jump' to a specific moment in the animation. Finally, Cinnamon et al. (2009) mention that the speed of the animation (pace) should be modifiable. This control is directly related to the temporal scale.

2.3 DECLARATIVE LANGUAGES FOR CARTOGRAPHIC VISUALIZATION

A programming language is a system of notation for describing computations (Tennent, 1981). In the context of the present research project, it is important to distinguish between imperative and declarative languages. Imperative languages aim to describe the computations step by step. It means, tell the computer *how* to do *what*, where *how* are the steps to produce the result *what* is expected. In opposition, with declarative languages the user specifies the intended result instead of the steps to accomplish it. In other words, the user specifies the *what* and let the computer decide *how* to do it. To clarify the difference between imperative and declarative languages, we provide the following example: Listing 2.1 presents a code written in a declarative language (SQL) and Listing 2.2 is its equivalent in an imperative language (JavaScript)(From Roberts (2013)).

Listing 2.1: Declarative code example - SQL query

```
SELECT * from dogs
INNER JOIN owners
WHERE dogs.owner_id = owners.id
```

Listing 2.2: Imperative code example - JavaScript equivalent of query in Listing 2.1

```
//dogs = [{name: 'Fido', owner_id: 1}, {...}, ... ]
//owners = [{id: 1, name: 'Bob'}, {...}, ...]

var dogsWithOwners = []
var dog, owner

for(var di=0; di < dogs.length; di++) {
  dog = dogs[di]

  for(var oi=0; oi < owners.length; oi++) {
    owner = owners[oi]
    if (owner && dog.owner_id == owner.id) {
      dogsWithOwners.push({
        dog: dog,
        owner: owner
      })
    }
  }}
}
```

From the example provided in Listings 2.1 and 2.2, the reader can notice the following differences:

1. The declarative code is shorter than imperative code.

2. It is easier to read the declarative code and therefore to understand the objective of it.

3. By using a declarative approach, the user can focus on the expected result and let the underlying software to deal with how to produce it.

We are not saying that declarative languages are better than imperative ones, neither the other way around. But under certain conditions it is better to use one or the other. In the particular case

of this project, we are interested in declarative languages, because they are easier to learn for non-programmers, and therefore it allow us to reach a broader community of users. In this regards, Heer and Bostock (2010) state that "the separation of specification from execution allows users to focus on the specifics of their application domain."

The visualization production systems can provide the user either with a Graphic User Interface (GUI) or a language to design the visual product. Regarding the ones using the language approach, Heer and Bostock (2010) indicated that most of them make use of the imperative programming model. This sets a big constraint in their use, because it implies that the visualization designer must have programming skills. However, there are several examples of declarative languages for data visualization as: Protovis (Bostock & Heer, 2009), D3 (Bostock et al., 2011), MapServer Mapfile (Lime et al., 2013), SMIL (W3C, 2012) and CZML (AGI, 2013b). All of them are capable to produce cartographic visualizations. Furthermore, SMIL and CZML can be used to create temporal animations.

Several declarative languages capable to specify animations were identified, but the ones capable to specify temporal animations operate at object level. Therefore, the production of an animation including a large number of elements is time-consuming. Among the languages that operate at set level, MapServer Mapfile presents the simplest syntax, so we decided to design a language for animation specification based on the MapServer Mapfile syntax.

2.4 SUMMARY

In this chapter, we described the visualization and analysis of dynamic phenomena by means of spatio-temporal animation. The principles and expected characteristics for animated maps were described. At the end of the chapter, a brief description of the usage of declarative languages for cartographic visualization productions was provided. The information presented in this chapter was used as the basis to design an approach for animated web map production based on a declarative language.

Chapter 3

Web animation out-of-the-box: an approach based on a declarative language

In this chapter, we propose an approach to overcome the deficiencies of current animation systems mentioned in Chapter 1. To provide a less time-consuming procedure for animated web maps production, we designed an approach based on a declarative language. The usage of a declarative language allows the user to focus on the specification of the animation, and let the system to take care of the production. Additionally, it reduces the code to be written by the user and simplify the understanding of the animation specification. The proposed language is capable to describe the access to regular data sources, hence the system is not constrained to work with a single specific data source. And regarding the performance of animations, we decided to include the option to specify the output format as part of the specification. In this way, the animated web map is not forced to be produced in a specific technology and there is always an option to implement new output formats that present performance or other functional improvements.

To provide a general idea of the proposed approach, Figure 3.1 shows a high-level architecture of the system, in which the general workflow can be explained as follows: the user writes an animation specification using the Animation SPECification Language or ASPEC-L. Following this, the toolset is executed, it analyzes the animation specification, retrieves the data from the indicated data sources and produces the required animation.

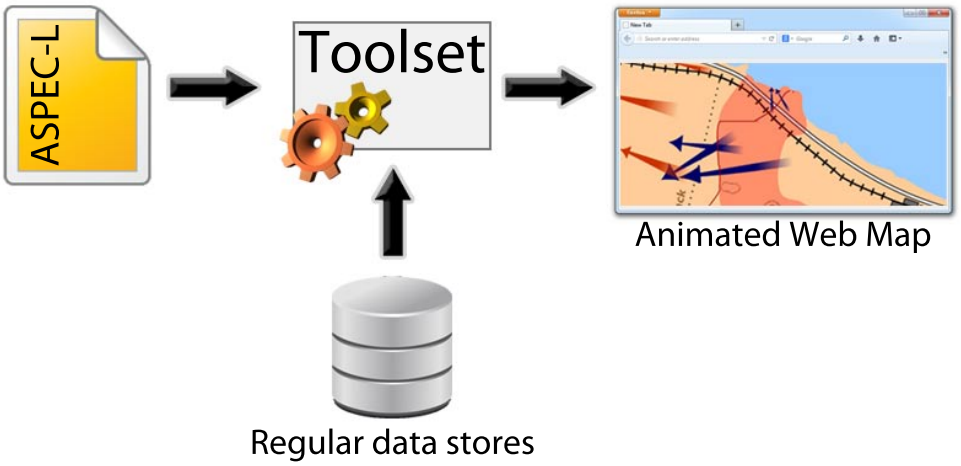


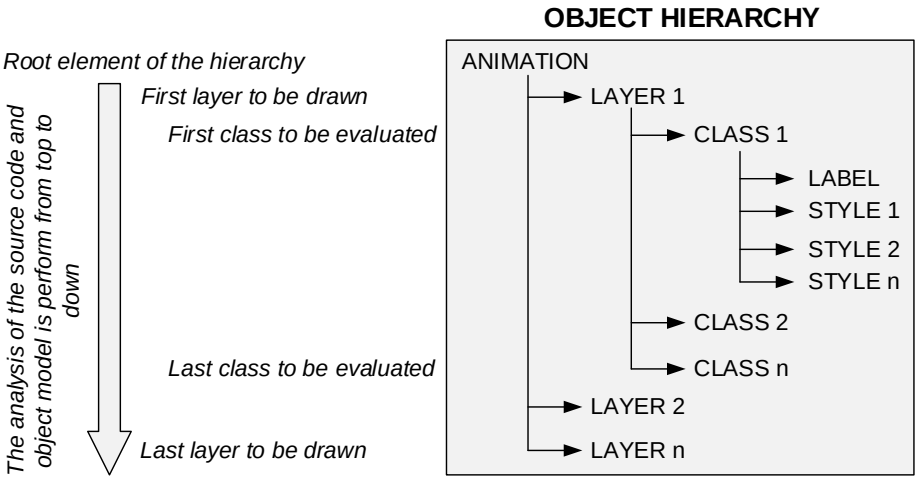
Figure 3.1: System high-level architecture.

This chapter is divided in two parts: the first describes the design of ASPEC-L, while the sec-

ond describes the design of a toolset that analyzes ASPEC-L code and produces animations from it.

3.1 ASPEC-L

ASPEC-L is a declarative domain-specific language (DSL) for animated map specification. This means that the user is capable with this language to specify 'what' is the output to be produced, instead of 'how' to produce it, and it is domain-specific, given that its application domain is the specification of animated maps. As this language is inspired by MapServer Mapfile syntax,[1] ASPEC-L code forms a hierarchy of objects to describe the content of the animated map. The general structure of an ASPEC-L file is shown in Figure 3.2.



This hierarchy defines several properties of the language regarding to the syntax and semantic of the language.

Regarding to the **syntax**, it defines that everything must be defined inside the **ANIMATION** object, it also defines which objects can contain which sub-objects, and the number of objects of each type that can be included.

Regarding to the **semantic,** it defines the order in which the layers are drawn in the animation, the order in which the classes are evaluated to select the proper one for the object being processed and the order to combine styles to create a composed one.

Figure 3.2: ASPEC-L object hierarchy.

The ***ANIMATION*** object is the root element of the hierarchy and within it, the complete definition of the animation is held. This object contains the properties that apply to the animation as whole and the ***LAYER*** objects that are included in the animation. The ***LAYER*** objects contain all the necessary information to access the data sources and with the help of ***CLASS***, ***LABEL*** and ***STYLE*** objects define how to visually represent the data in the animation.

[1]Full description of MapServer Mapfile is available in http://www.mapserver.org/mapfile/

3.1.1 Design process

The design of the language was guided by three important questions: how to specify the animation to be produced?, how to define the data to be used?, and how to set the visual representation of the features included in the animation? By answering these questions the attributes to describe an animated map were defined.

To answer the first question, it was necessary to start by defining the types of animated maps to be specified. Our interest is to have the capability to specify animations to represent data time series in raster and vector domains. With these types of animation, it is possible to depict existential, spatial and attribute changes over time. In the raster domain, basically, animations are built by showing a sequence of images one after another, therefore the language allows to define an attribute to specify how to perform the change between images. In the vector domain, the description of the animations is more complicated. Existential changes are depicted by features that appear, disappear or a combination of both, separated by a time lapse, and spatial and attribute changes are depicted by variations of location, shape and visual representation of the features. Additionally, an existential change can occur when the phenomenon starts or ends, and the features have to (dis)appear. Depending on the temporal sampling rate of the data, when the changes are represented in animation time, the animations may appear abrupt. Nevertheless, one of the most distinctive characteristics of animated maps is that they can show smooth transitions. For this reason, ASPEC-L allows to specify whether the spatial and attribute changes should be represented in stepwise mode or in interpolated mode.

The depiction of spatio-temporal phenomena requires to consider both the spatial and temporal dimension. In the spatial dimension, the user may want to specify the spatial extent in which the phenomena under study take place, and in the temporal dimension, the language allows to specify the real-world time span to be represented and the duration of the animation. Additionally, one needs to indicate the data to be used and the visual representation to be applied.

Answering the second question, we are interested to consume data from files, OWSes and spatial databases. To describe the access to these data sources, various parameters, must be includes: the type of data source, the location of the data source, the name of the dataset in the data source and the type of data that it contains. In addition, for spatial databases it is necessary to specify the connection string to the server and for OWSes the URI (Uniform Resource Identifier) of the service. It is expected that in many cases the user will need to use a subset of the dataset, in this sense the time span to be represented and the spatial extent to be depicted, provide temporal and spatial filtering, but additionally, an option to specify a filter based on thematic attributes is included. Another important consideration regards the dataset structure, as it changes from one dataset to another, it is necessary for the language to allow the user to specify the name of the fields in which the geometries and timestamps can be found; Finally, when working with spatial data, one needs to allow different spatial reference systems (SRS), for this reason the language allows the specification of a SRS for inputs and outputs of the system.

While considering the last question, we decided to include in the language the following visual properties: fill color, outline color, outline width, symbols, size of symbols and labeling. Additionally, expressions based on the thematic attributes can be used to select subsets within a dataset to apply different visual styles. As the values of the thematic attributes may change over time, this is translated in the animation as re-expression. Which means that the animation represents the changes in attributes by changing the visual representation of the objects over time.

Once the attributes to describe the animations are defined, they are organized in objects to define their scope, which defines the hierarchical structure shown in Figure 3.2. Before describing the objects and properties in detail, it is necessary to describe data types and expressions.

3.1.2 Data types

The data type of an attribute defines the values that can be assigned to it. A wrong data type assignation will lead to an error when analyzing the code or producing the animation. The valid data types in ASPEC-L are described in Table 3.1.

Table 3.1 Data types in ASPEC-L

TYPE	EXAMPLE	DESCRIPTION
BOOLEAN	True	Data type that represents truth values, it can be True or False.
INTEGER	180	Number without decimal part.
DECIMAL	292.58	Number with decimal part.
STRING	"city = 'Enschede'"	String of characters that can include letters, numbers and special characters. All string values are enclosed in quotation marks.
COLOR	125 90 0 100	A sequence of 4 integer values. The first 3 represent the red, green and blue component of the color and ranges from 0 to 255; and the last is the opacity of the color, which ranges from 0 to 100. Value 0 indicates completely transparent and 100 completely opaque.
EXTENT	-10.0 20.5 15.25 60.0	A sequence of 4 decimal numbers, which represent a spatial extension contained on the bounding box specified as "minX minY maxX MaxY".
TIME	20131001T12:35:00	Represents a date/time value in compliance with ISO8601:2004.
DURATION	01:30	Represent a duration in time with the format mm:ss, where mm indicates the number of minutes and ss the number of seconds, both values should be within the range 0 and 59.

3.1.3 Expressions

In programming, a general definition for expression is a combination of literal values or constants, variables, operators and functions that can be interpreted to produced a result. The data type of the result depends on the operands and operators included in the expression. ASPEC-L has two types of expression: arithmetic expressions, which produce numeric results, and logical expressions, which produce booleans. In ASPEC-L the expressions can include literals, field names and operators. The valid operators are described in Table 3.2, for simplicity we use A and B to refer to literals or field names.

3.1.4 Lexical elements

The lexical elements of a language define the words that form part of it. The set of lexical elements for ASPEC-L are keywords to define objects and properties, property values and comments. The

Table 3.2 Operators in ASPEC-L

ARITHMETIC OPERATORS		
OPERATOR	**EXAMPLE**	**DESCRIPTION**
+	A + B	Addition: Calculate the sum of A and B.
-	A - B	Subtraction: Subtract B from A.
*	A * B	Multiplication: Multiply A times B.
/	A / B	Division: Divides A into B.
%	A % B	Modulus: Returns the remainder of the division of A into B.
**	A ** B	Exponent: Calculate A to the power of B.
//	A // B	Integer division: Divides A into B, ignoring the decimals in the result.
COMPARISON OPERATORS		
OPERATOR	**EXAMPLE**	**DESCRIPTION**
=	A = B	Equal: Compares A and B, if both are equal the result is True.
!=	A != B	Not equal or different: If A is not equal to B, then the result is True.
>	A > B	Greater than: The result is True, when A is greater that B.
>=	A >= B	Greater or equal: The result is True, when A is greater than or equal to B.
<	A < B	Less than: The result is True, if A is less than B.
<=	A <= B	Less that or equal: The result is True, if A is less than or equal to B.
LOGICAL OPERATORS		
OPERATOR	**EXAMPLE**	**DESCRIPTION**
AND	A AND B	Logical And: Returns True if A and B are true.
OR	A OR B	Logical Or: It returns True if at least one operand is true.
NOT	NOT A	Logical Not: It returns the opposite value of A, so it returns True if A is False.

comments in ASPEC-L are defined as free formated text that starts with a hash symbol (#) and finishes with an end-of-line character. Comments have no influence on the animation to be generated, but they provide the user with a mechanism to make annotations within the animation specification.

In ASPEC-L, there are five keywords used as opening for object definition: ***ANIMATION***, ***LAYER***, ***CLASS***, ***LABEL*** and ***STYLE***. Every object has several properties and for each property, there is a keyword. For a full list of keywords for properties the reader is referred to Appendix A. Additionally, the ***END*** keyword is used to finalize any object definition.

Finally, in the case of property values, we can distinguish among two cases: string constants that are strings with special meaning for a particular attribute, and values that should match a type/format specification. The string constants and attributes in which they are applicable are described in Appendix A, and for type/format specification we refer to Table 3.1.

3.1.5 Language syntax

Valid code written in ASPEC-L forms a hierarchy of objects. In this hierarchy the ***ANIMATION*** object is the root element, and within it, all the properties and layers that form the animation are specified. An object is defined by using opening and closing keywords. It may contain properties and sub-objects. The order in which the attributes are defined within an object have no

importance, and the sub-objects that can be included within an object definition are defined by the hierarchy shown in Figure 3.2.

The end of statement is indicated by the end of line, so each line can only contain one object or property definition. However, comments are allowed in the same line with the statement. Empty lines, extra spaces at the start and end of line or in between of the keywords and property values are ignored.

Finally, the language is not case-sensitive, but it is important to take into account that the field names and values for filtering and classification may need to be case-sensitive depending on the data source.

3.1.6 ASPEC-L sample code

The example of Listing 3.1 is provided to complement the explanation for the language lexical and syntactical elements.

Listing 3.1: ASPEC-L sample code.

```
ANIMATION
    NAME "Lilac blooming 1968"
    START_TIME "1968-01-01T00:00:00"
    END_TIME "1968-06-30T00:00:00"
    DURATION 01:00
    FORMAT "WEBGL"
    BBOX -124.36 29.53 -52.78 49.25
    BBOX_BEHAVIOUR "FIXED"
    OUTPUT_NAME "lilac1968"

    #Vector layer
    LAYER
        DATASOURCE "WFS"
        TYPE "POINT"
        DATA "lilac"
        URL "http://www.some-ows.nl/"
        ANIMATION_TYPE "EXISTENTIAL"
        FIELD_TIME "bloom"
        DURATION 2
        CLASS
            STATUS "ON"
            STYLE
                COLOR 255 0 0 100
                SIZE 6.0
                OUTLINE_COLOR 0 0 0 100
                OUTLINE_WIDTH 2
            END
        END
    END

    #Raster layer
    LAYER
        DATASOURCE "WMS"
        TYPE "RASTER"
```

```
35          DATA "temperature"
36          URL "http://www.some-ows.nl/"
37          ANIMATION_TYPE "DIRECT"
38      END
39  END
```

3.2 TOOLSET DESIGN

Up to this point, we described the design of a declarative language that provides the user with the ability to specify animated maps, the so-called ASPEC-L. However, the aim of this research project, is not just to describe the animations, but to produce them. Therefore it is necessary to design a mechanism capable to convert such specifications into animations, and this functionality is provided by a toolset.

The toolset was designed on the basis of the architecture of a generic compiler. Louden (2004) defines a compiler as computer software that is capable to translate code in one language to another, where the input is called source code and the output object code. The object code is an equivalent representation of the source code, but this one can be executed on some target platform. A high-level representation of this generic architecture is shown in Figure 3.3. The source code is analyzed by a software component that is called the front-end, it produces an intermediate representation of the code, that is not aimed to be executed by any platform, but that simplifies the task of another software component that is called the back-end, which produce the object code. This architecture has the advantage that several back-ends can be developed for the same front-end, therefore the object code can be produced for different platforms. This characteristic is important for us, because we want to keep open the possibility of produce different output formats.

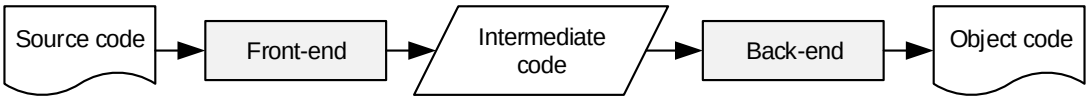


Figure 3.3: Architecture of a generic compiler. Translated from Louden (2004).

We designed a toolset that uses ASPEC-L to produce animated web maps. Figure 3.4 shows the architecture and the internal workflow of the toolset. The toolset architecture is divided in three components: *source code analyzer*,[2] *animation producer* and *animation renderer*. The first two are executed on the server-side[3] and the third one on the client-side. The back-end is represented in the toolset by the *animation generation*. It implies that if we want to produce different output formats, we just need to develop different implementations of that component. A detailed description of the components and workflow of the toolset is given in the remaining sections.

[2]Source code in reference to ASPEC-L files.

[3]Not necessarily the same server in which the animation is published, but this separation is useful to indicate that they are executed somewhere else that is not the client machine.

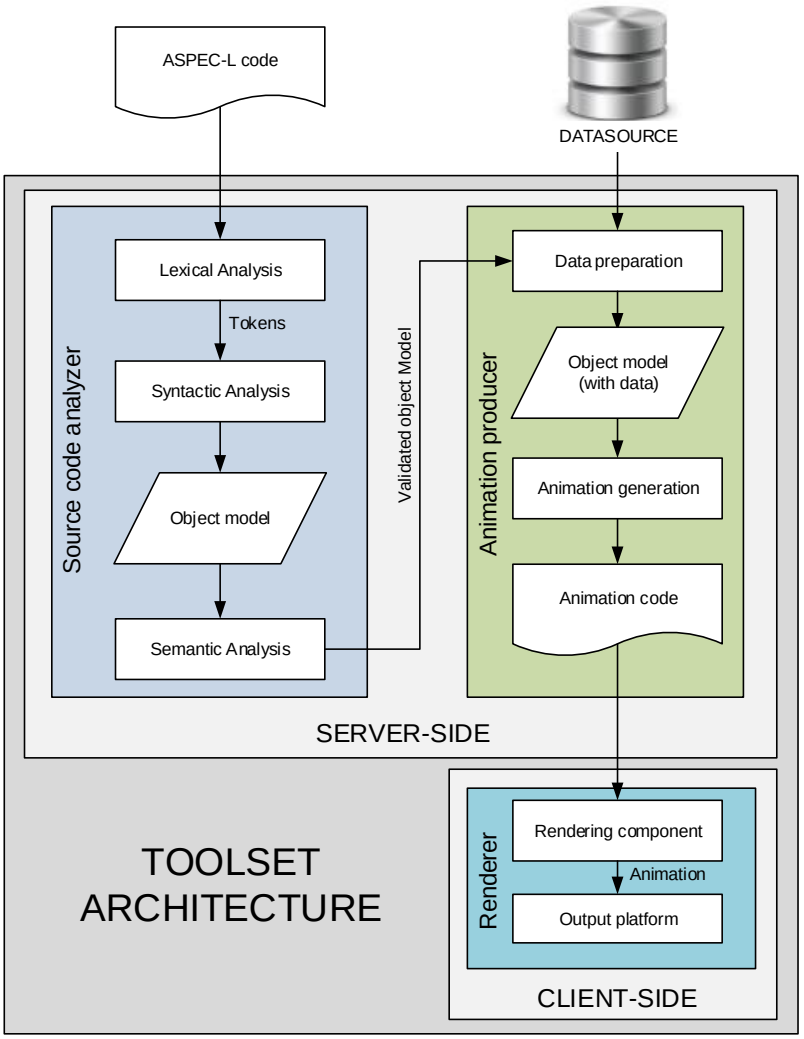


Figure 3.4: Toolset architecture and internal workflow.

3.2.1 Source code analyzer

This component receives the ASPEC-L code as input and produces a validated object model as output (see Figure 3.5). To achieve this, the source code is analyzed and the proper objects are instantiated to generate an object model. Once the object model is built, it is necessary to analyze it, to determine whether the specified animation can or cannot be built.

Lexical analysis

The *lexical analysis* consists of splitting the text into lexical elements and transfer them to the *syntactic analysis* as tokens. Tokens are structures that store the type of lexical element and the value of it. For instance a token of type 'Object definition' with the value 'ANIMATION'. This task is accomplished by concatenating individual characters to form the longest possible strings that match with one of the patterns that define the lexical elements of the language (see Figure 3.6). The lexical elements for ASPEC-L can be consulted in Appendix A.

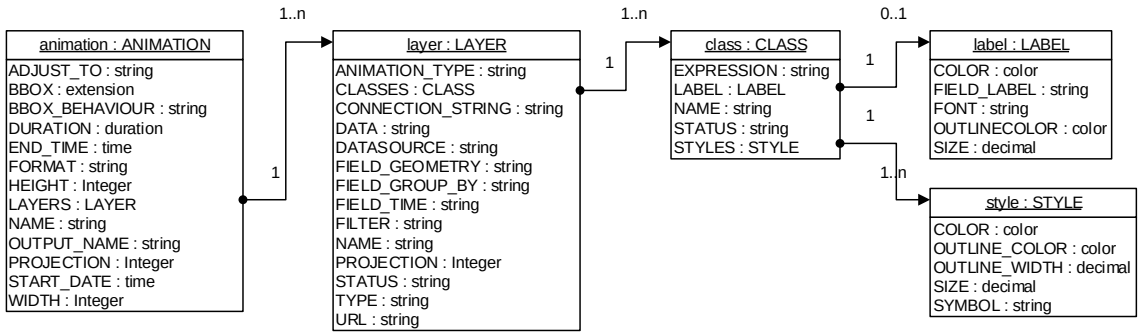


Figure 3.5: ASPEC-L object model.

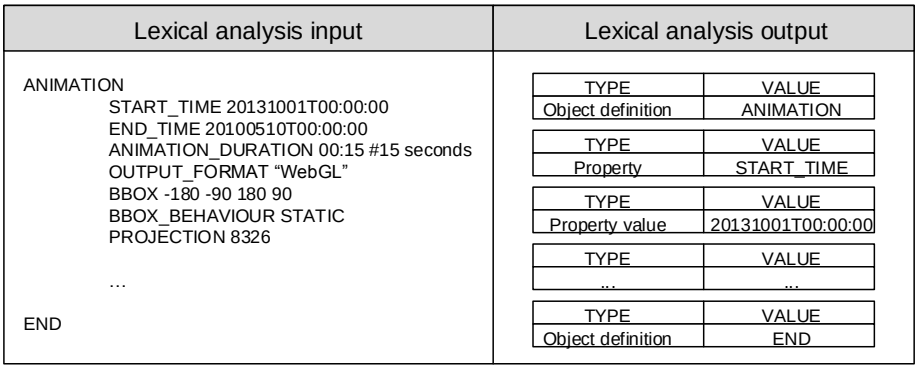


Figure 3.6: Production of tokens.

Syntactic analysis

The aim of this stage is to ensure that the tokens form a valid structure based on the language syntax presented in Section 3.1. For our case, validity means that:

1. No properties or objects are declared out of the ***ANIMATION*** object
2. Properties are declared within the appropriate object
3. The values for the properties are of the correct data type
4. Sub-objects are defined within an object that can contain them

The syntactic analyzer receives a token stream and produces an object model (see Figure 3.5). The produced model is valid from the point of view of object model structure, but it doesn't imply that it describes an animation that can be produced.

Semantic Analysis

The semantic analysis aims to validate the object model. This validation is performed to determine if the object model describes an animation that can be built. The first step in this validation process is performed for all objects. It consists of checking if all the compulsory fields were specified or not.

On an ***ANIMATION*** object, it is necessary to check: if the ***PROJECTION*** attribute contains a valid EPSG code, this specifies the SRS in which the output will be produced. The time span should be valid, it means that the starting time should be before the ending time and both should be specified in compliance with ISO8601:2004. If ***BBOX_BEHAVIOUR*** is set to ***FIXED*** or ***USER***, ***BBOX*** should be defined. And the dimensions of the output specified by ***HEIGHT*** and ***WIDTH*** attributes should be greater than zero.

In regards to ***LAYER*** objects, they all should properly define the access to the data source. The ***PROJECTION*** attribute must contain a valid EPSG code, this specifies the SRS in which the data is provided. For ***EXISTENTIAL*** animation, ***FIELD_TIME_2*** or ***DURATION*** must be specified. If the animation to be produced is of type ***STEPWISE*** or ***INTERPOLATED***, ***FIELD_GROUP_BY*** must be specified. Finally, if the data source specified contains more than one geometric field, the ***FIELD_GEOMETRY*** attribute should be defined.

Finally, each ***LAYER*** object should include at least one ***CLASS*** with a ***STYLE*** that defines enough attributes to be visible in the animation, and ***LABEL*** objects may be or not defined.

3.2.2 Animation producer

This component performs two tasks: *data preparation* and *animation generation*. The input for this component is a valid object model and it produces as output the animation. Despite the fact that all the required parameters are specified correctly on the object model, it is still possible that the data sources are not available when trying to create the animation or that the data is not valid. If this happens, the animation cannot be produced.

Data preparation

This stage aims to prepare the data in a generic format to facilitate the production of the animation. The data preparation is divided into vector and raster data preparation. The workflows for these are depicted in Figure 3.7. From the workflows, it can be thought that an interpolation stage is missed, but we decided not to include such, because the need of it is output format dependent. Therefore, if interpolation is needed it takes place in animation production.

The first process on the vector data preparation workflow is to obtain the data from the data source. To accomplish this task, the toolset creates an instance of the proper module to work with the specified data source and retrieves the data from it. At this stage the data is filtered by time, and if no time span is defined, the whole dataset is loaded. A temporal local table is created with this data. The geometric field should be stored in the temporal local table in a standard format, as for example GeoJSON (Geo JavaScript Object Notation) or WKT (Well Known Text). This simplifies the work with the geometries in the following processes.

The *attribute filtering* process filters records based on their attribute values. The criteria for this filtering process are specified in the attribute ***FILTER*** of the ***LAYER*** objects. Whenever possible, this filtering stage, should be performed in combination with the *load vector data* process, to avoid loading unnecessary data.

Different datasets store the timestamps in different formats. Therefore, it is necessary to convert all the timestamps to a single format. Nevertheless, there are so many formats to represent timestamps, that it is not realistic to work with all of them. To overcome this drawback, the toolset

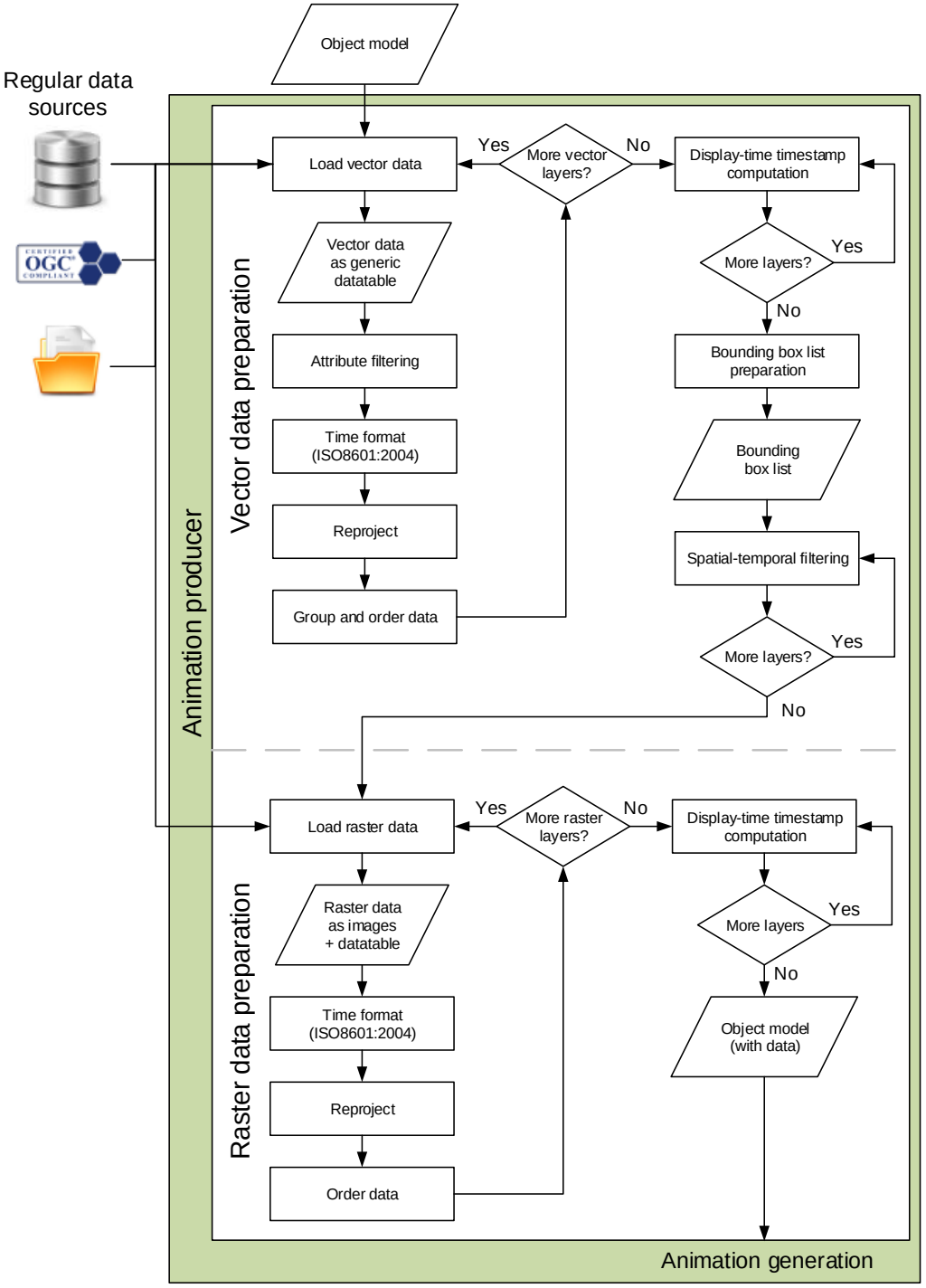


Figure 3.7: Data preparation workflow. Including vector and raster data preparation.

is designed to work with ISO 8601:2004 standard compliant timestamps. A summary of the most common date/time formats is presented in Table 3.3. After formatted, the timestamps are represented in YYYY-MM-DDThh:mm:ss format (e.g. 2013-09-16T12:30:10).

Table 3.3 ISO 8601:2004 date/time formats.

REPRESENTS	FORMAT	EXAMPLE
DATE	YYYYMMDD	20130120
	YYYY-MM-DD	2013-01-20
	± YYYYYYMMDD	+0020130120
	± YYYYYY-MM-DD	+002013-01-20
	YYYYDDD	2013020
	YYYY-DDD	2013-020
TIME	hhmmss	123045
	hh:mm:ss	12:30:45
DATE/TIME	YYYYMMDDThhmmss	20130917T123045
	YYYY-MM-DDThh:mm:ss	2013-09-17T12:30:45
	± YYYYYYMMDDThhmmss	+0020130917T123045
	± YYYYYY-MM-DDThh:mm:ss	+002013-09-17T12:30:45
	YYYYMMDDThhmmss±hhmm	20130917T123045+0100
	YYYY-MM-DDThh:mm:ss±hh:mm	2013-09-17T12:30:45+01:00

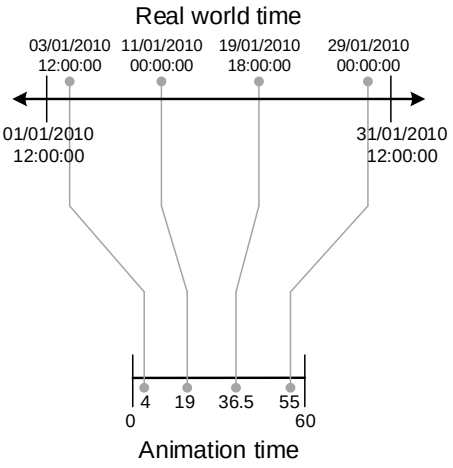
Just as with timestamps, geometries may be stored in different SRSes. The re-projection phase, transforms the geometries to the SRS in which the animation will be produced. This ensures that all objects will be properly represented in the spatial canvas. If the dataset is already in the proper projection, this phase is skipped.

As the toolset produces a temporal animation, it is necessary to have the data chronologically ordered and depending on the type of animation, be grouped as well. To achieve this, the data is grouped by the field specified in ***FIELD_GROUP_BY*** and ordered by the field specified by ***FIELD_TIME***.

Depending on the output platform, it is possible that display-time timestamps are required instead of ISO 8601:2004 compliant ones. By display-time timestamps, we mean that the timestamps are expressed relative to the animation duration. This stage can only take place after all layers have been prepared, this is because when the time span of the animation is not provided, it should be calculated from the time span covered by the layers. Once the timespan of the animation is known, the process to calculate the display-time timestamps is as follows: first, the temporal scale is calculated, this is done by subtracting the end time from the start time, and dividing the result by the duration of the animation; and second, the start time is subtracted from the timestamps and the result is divided over the temporal scale. This procedure is illustrated in Figure 3.8.

The *bounding box list preparation* stage calculates the spatial extents that will be depicted in the animation. This list will contain values indicating when to change the spatial extent and which area should be displayed. This computation depends on the ***BBOX_BEHAVIOUR*** attribute. If this attribute is set to ***AUTO***, the toolset computes a series of bounding boxes, to follow the development of the phenomenon. Otherwise, the list will just include the starting time with the spatial extent indicated in the specification file.

The spatio-temporal filtering stage discards those objects that will not be displayed in the an-



Formulas to calculate the display-time timestamps are:

$$TSF = \frac{(ET - ST) \times 86400}{AD} \qquad AT = \frac{(T - ST) \times 86400}{TSF}$$

Example using:

Timestamp 11/01/2010 00:00:00

Animation duration 60 seconds

$$TSF = \frac{(31/01/2010\ 12{:}00{:}00 - 01/01/2010\ 12{:}00{:}00) \times 86400}{60} = 43200$$

$$AT = \frac{(11/01/2010\ 00{:}00{:}00 - 01/01/2010\ 12{:}00{:}00) \times 86400}{43200} = 19$$

ST = Start Time — TSF = Time Scale Factor

ET = End Time — T = Timestamp

AD = Animation Duration — AT = Animation Time

Notes:

1. *The value 86400 is a constant and represents the number of seconds in 1 day.*
2. *When extracting one timestamp from another, the result is the number of days between them.*

Figure 3.8: Procedure to compute display-time timestamps.

imation, this process can be described as a spatio-temporal intersection between the objects and display area. Spatio-temporal filtering takes place only if the user cannot change the spatial extent during playback, because otherwise there is no certainty if an object will be displayed or not. The objects can be within the displaying area, can be partially on it, or can be completely outside of it, and this status can change over the time. The objects that can be discarded are those which will not be displayed and are not needed for interpolation. This stage is particularly important for the performance of the animation, because it will avoid, for the rendering component, computations of objects that will not be displayed. Figure 3.9 illustrates the spatio-temporal filtering on point data for interpolated movement, this is a simple example assuming a fixed spatial extent. The generalization of this concept requires to consider: types of animation, data types and the behavior of the spatial extent.

Once vector data preparation is completed, the next step is to prepare the raster data. By instantiating the proper component to work with the specified data source and using the bounding box list, images are prepared from the raster data source and they are stored in a temporal local folder. Besides, a table with the timestamps, extent covered and image paths should be created. An important consideration is that the images are prepared for a specific spatial scale. Therefore the options for panning and zooming are not available when using raster animation. Otherwise, preparation of raster data should be performed during playback. The time formating, reprojection, ordering data and computation of ordinal timestamps follows the same logic as explained for vector data.

Animation code generation

At this stage, the toolset instantiates the proper module to produce the animation in the specified format. The general workflow of this phase is depicted in Figure 3.10. The production process is format-dependent, therefore no specific details of the implementation of each format are provided here. However, the general approach to produce the animation based on the type of data

and change to be depicted are described in the following paragraphs and are illustrated in Figure 3.11.

Three types of raster animation can be produced: direct change, it means that the current image is directly replaced by a new one, based on the timestamps. Faded change, the current image is faded out, while the new images is already shown, this change should be synchronized in such a way that the old image is completely faded out by the time the new one must be visible, based on timestamps. And interpolated change, in this case intermediate images are computed by means of interpolation and then the animation is created as direct change.

In the case of vector data, the types of change that can be depicted are: existential, spatial and attribute. When creating an animation to depict an existential change, ***FIELD_TIME*** defines when the features appear and ***FIELD_TIME_2*** when they disappear. Alternatively, the ***DURATION*** attribute specifies the number of seconds in animation time that the features are displayed. The animations to depict spatial (location and shape) and attribute changes can display them in stepwise or interpolated mode. To create these animations, the data is grouped in objects, therefore several states for each object are available. Creation of an animation in stepwise mode implies that just the macro-steps are shown. Each state is displayed from its timestamp value to the timestamp of the next state. In the interpolated representation, the changes are smoothed by adding micro-steps, to calculate the micro-steps interpolation is used. If the output platform is capable of computing interpolation, this process can take place on-the-fly during playback. This type of functionality is present in Flash (Adobe, 2014) and SVG+SMIL (Becker, 2009) platforms. Nevertheless, we have to consider that this is a computationally-expensive procedure that negatively affects the playback performance. Another option is to compute the micro-steps during the animation production and produce the animation as stepwise animation.

3.2.3 Renderer

This component plays the animation. The rendering component is a web application or a plugin capable to display the animation in the output platform. Given that the system produces animated web maps, the output platform is a web browser.

Depending on the output format and the implementation of it, the user is capable to control the animation, with a selection of tools that may include:

1. *Basic control*: Provides the user with the capability to play, resume and stop the animation. It is recommended that the produced animations, include at least this level of control.
2. *Time control*: The user can change the speed, the playback direction and the current moment being playing of the animation.
3. *Spatial control*: Allows the user to change the spatial extent of the visualization, for this aim controls are provided for pan and zoom.
4. *Querying control*: To better understand the animation, the user may require to query the data on it. Two mechanisms are foreseen to implement this control: by clicking on the animation, the user get a table containing the information of the selected objects; or, the user can select from a table the objects of interest, and the result is displayed in the animation by highlighting the selected objects or by hiding the non-selected ones, in which case it works as an interactive filter.

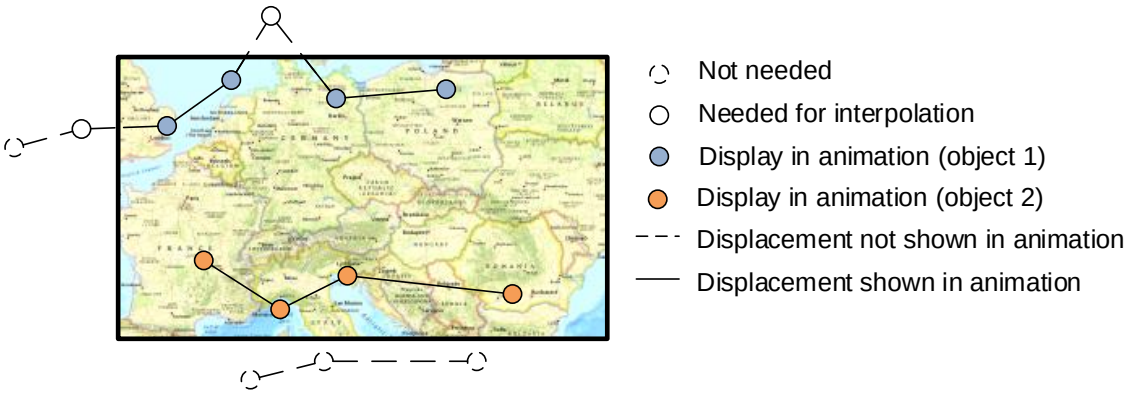


Figure 3.9: Spatio-temporal filtering on point data.

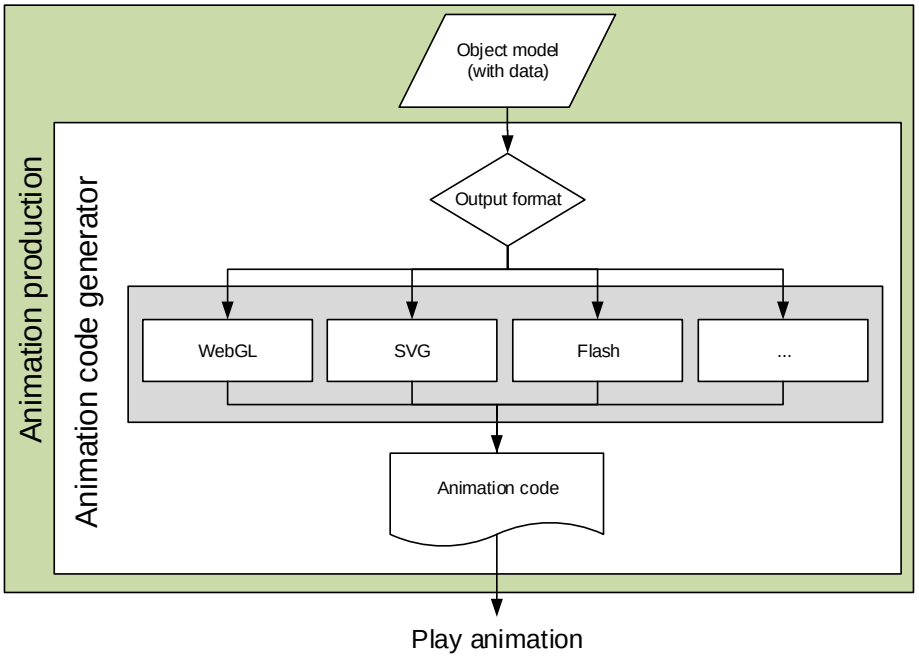


Figure 3.10: Animation code generation workflow.

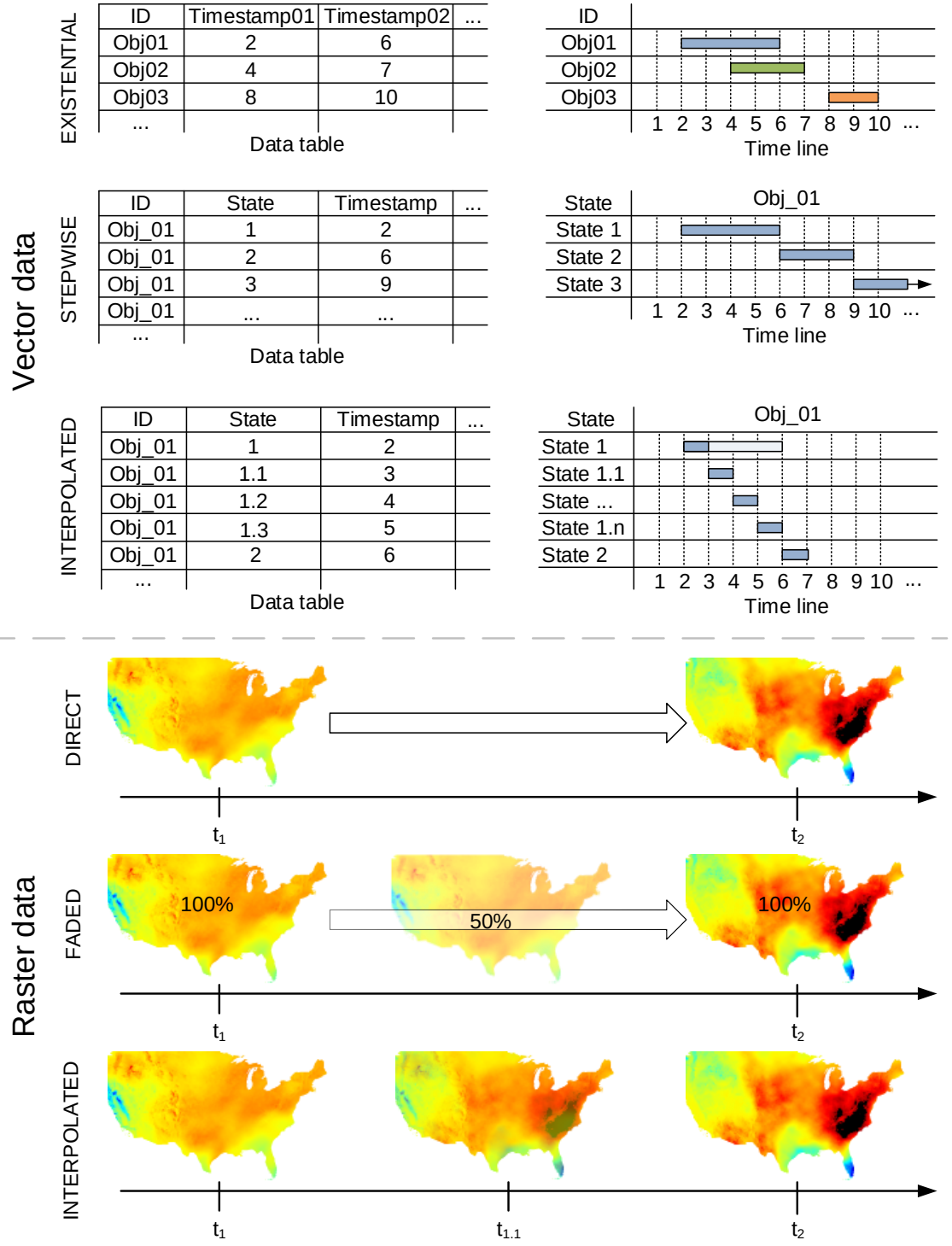


Figure 3.11: Generic procedure to produce animation.

3.3 SUMMARY

In this chapter, we presented an approach to produce animated web maps based on a declarative language. The Animation SPECification Language or ASPEC-L, is a declarative domain-specific language inspired in MapServer Mapfile syntax, it was designed to provide a systematic and consistent procedure to specify animated web maps. A toolset was designed as mechanism to produce the animations from the specifications written in ASPEC-L. Both the language and the toolset were designed to allow further extensions.

Chapter 4
Prototype implementation

This chapter describes the implementation of a prototype for the design presented in Chapter 3. An incremental approach was used to develop the prototype. The idea behind this approach is to develop a first version of the software with minimum functionality and add characteristics in subsequent steps. The use of this approach ensured that by the end of the project, a not complete, but functional system was developed. A five steps plan was prepare to develop the prototype, the planned steps are shown in table 4.1, from the original plan, the first four steps were accomplished.

Table 4.1 Prototype development plan.

			Characteristics			
		Language version	**Raster layer**	**Vector layer**	**Controls**	**Extent behavior**
Step	1	0.2	Static	Animated	No controls	Static
	2	0.4	Static	Animated	Play and stop	Static
	3	0.6	Static	Animated	Play, stop, go forward and backward in time	Static
	4	0.8	Static	Animated	Play, stop, go forward and backward in time	Dynamic
	5	1.0	Animated	Animated	Play, stop, go forward and backward in time	Dynamic

To simplify the development of the prototype, we reduced the ASPEC-L specification to ***ANIMATION***, ***LAYER*** and ***CLASS*** objects. The attributes of ***STYLE*** object were merged with ***CLASS*** object, which implies that only one style definition is allowed per ***CLASS*** object, and that the ***LABEL*** object was not implemented. The class diagram for the implemented object model is shown in Figure 4.1. The workflow of the toolset was reduced as well. The display-time timestamps computation and spatial filtering stages were not implemented. Figure 4.2 illustrates the implemented workflow. These modifications simplified the coding of the prototype, without affecting the essentials of the approach.

In the language definition, we simplified the mechanism to specify the visual properties of the objects in the animation. Which simplifies the development of the object model, the source code analysis and the animation production, but sets a constrain in the visual representation of the objects, to the usage of simple styles. We don't foreseen any challenges in the modifications required on the object model and source code analysis to implement the ***STYLE*** and ***LABEL*** objects. Nevertheless, the inclusion of this features in the animation production is format-dependent, and will represent an extra work load for the rendering component, therefore, the animation code must be

optimize to reduce the negative impact on the playback performance.

In the development of the prototype, the display-time timestamps computation was not implemented, because the library used in the rendering component can work directly with ISO 8601:2004 timestamps. The implementation of this feature can be done using the *datetime* object of Python. Regarding to the spatial filtering, it was not implemented because it is an optimization process that is not compulsory for the animation production, but that is important to optimize the playback performance. We foreseen that the inclusion of this feature in the toolset will negatively affect the animation production performance, while improving the playback performance. This is due to the extra computations required to filter the objects that will lead to the production of an optimized animation code.

The prototype in its current state of development can produce existential and stepwise animation for point, line and polygon data. Additionally, for point data, interpolated animations can be produced, which can include interpolation of position, size and color. The prototype consumes the data from one or more WFSes and produces the animated map in WebGL.

4.1 TECHNICAL CHOICES

To develop the prototype a variety of tools were chosen. They include a programming language, specific libraries to work with general and spatial data and JavaScript libraries. The prototype is composed of two applications: the animation production software and the rendering application.

The animation production software was built using Python. This decision was made on the basis that Python is a general purpose language, is easy to learn, increases development speed and no proprietary software is required (Python Software Foundation, 2013). Python includes functionality to work with data structures, but to facilitate the manipulation of general and spatial data, three libraries were selected, they are: Pandas, an open source library, which offers high performance, easy-to-use data structures and data analysis tools (The pandas development team, 2013); and two libraries, GDAL for manipulating geospatial raster data and OGR for manipulating geospatial vector data (Open Source Geospatial Foundation, 2013).

The rendering component was built as a web application. To built this component, we used HTML5 (HyperText Markup Language version 5), CSS3 (Cascade Style Sheet version 3) and JavaScript. With the aim to accomplish a good performance of animation playback, we decided to use WebGL to render the graphics. WebGL is a cross-platform, royalty-free API, capable of producing hardware-accelerated 2D and 3D graphics for web browsers (Khronos Group, 2011). Given that WebGL is a low-level API, we decided to look for a library that provides an intermediate layer between our application and WebGL. CESIUM was identified as the most suitable library for our needs. This library offers four levels of abstraction as shown in Figure 4.3. The abstraction levels range from the *core* that provides low-level functionality for 2D and 3D graphics, to the *dynamic scene* which makes it possible to build data-driven animations based on CZML. CZML is a declarative language to describe temporal-animations at object level (AGI, 2013a).

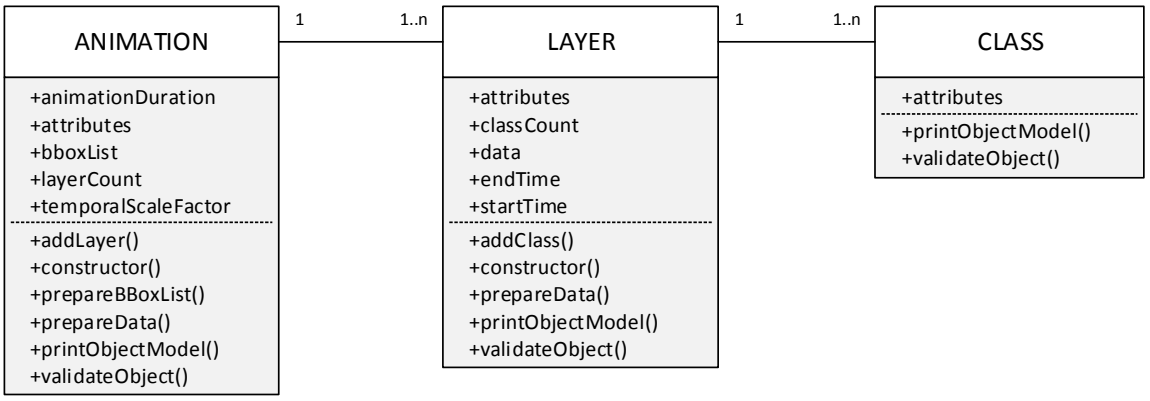


Figure 4.1: Class diagram for the object model.

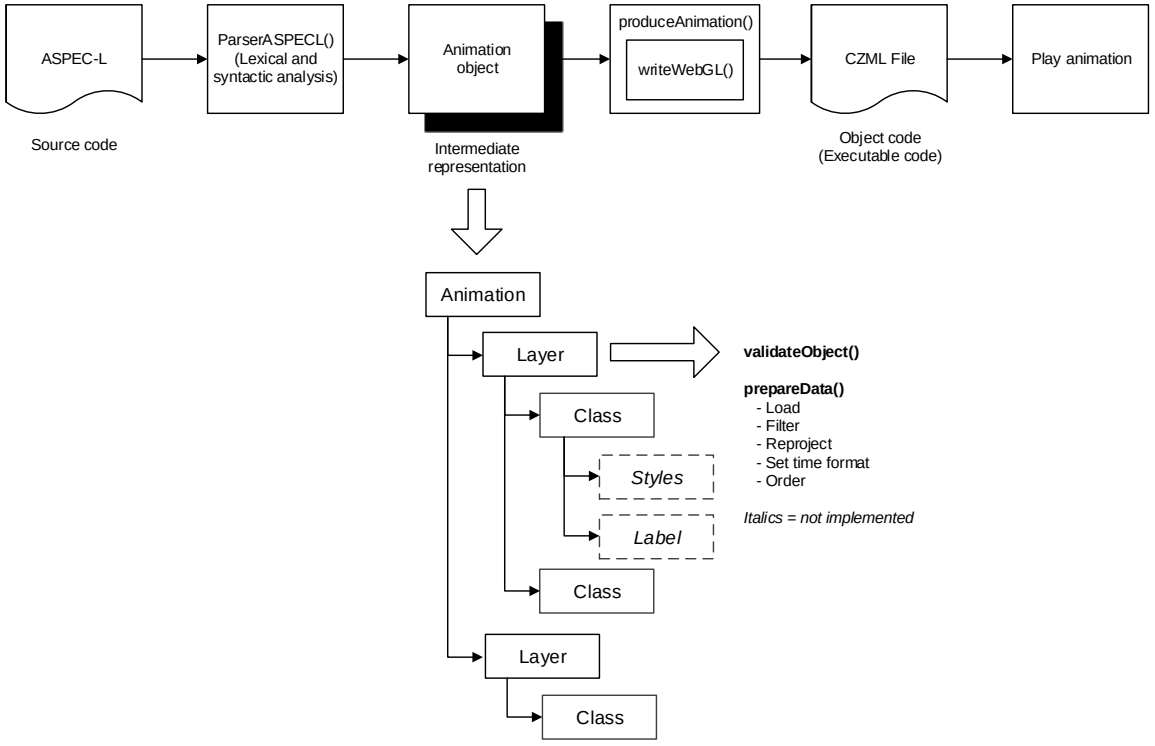


Figure 4.2: General workflow implemented in prototype.

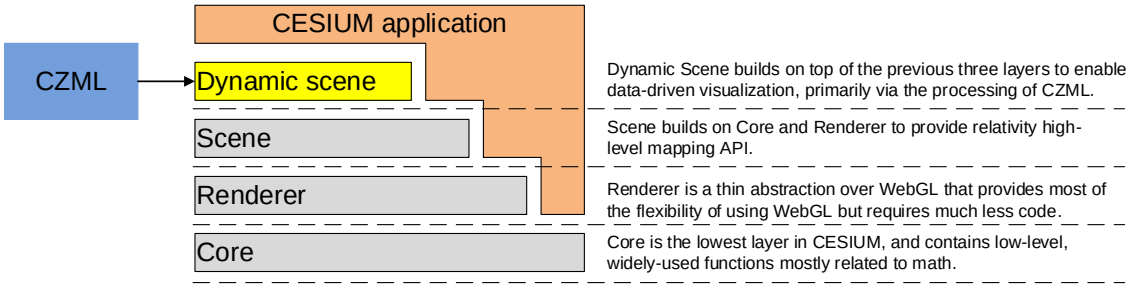


Figure 4.3: CESIUM high-level architecture. From AGI (2013a).

4.2 ANIMATION PRODUCTION SOFTWARE

Two components of the toolset are included in this software, the source code analyzer and the animation producer. The general workflow is presented in Algorithm 1. Every task of the workflow was implemented as a function, which returned value is used to determine if it was executed successfully or not. When an error arises, it is reported and the system finalizes without producing the animation. Otherwise, the output is the animated web map ready to be disseminated.

4.2.1 Source code analysis

The lexical and syntactical analysis are combined in a function called *parserASPECL*, as shown in Algorithm 1. This function is implemented as a Deterministic Finite Automaton (DFA). The design of the DFA is depicted in Figure 4.4. At each state, one or more tokens are generated by getting a line from the source code and separating it into *keyword* and *value*. If the line contains an object opening or closing keyword, *value* is returned as empty string. Empty lines and those containing only comments are discarded. In a DFA, the state transitions are fully determined by the current state and the inputs (tokens) received. Any state transition not depicted in the figure, leads to an error state and terminates the analysis of the ASPEC-L code, therefore the animation will not be produced. On the other hand, if there are no more lines in the source code to be analyzed and the DFA is in "OBJECT MODEL" state, the code was successfully analyzed and the object model is built. Algorithm 2 presents the implementation logic of the DFA.

The implementation of the semantic analysis was done by adding a *validateObject* method on every object (See Figure 4.1). The validation process is based on the description provided in Chapter 3. To perform the validation, the *validateObject* method on the ***ANIMATION*** object is called and this propagates the call to the other objects. If an object cannot be validated, the *validateObject* method returns False and the result is propagated back, the error is reported and the code analysis terminated. If no error occurs the object model is valid and the production process can continue.

4.2.2 Data preparation

This stage can be divided into four tasks: vector data preparation, computation of the temporal scale factor, preparation of the bounding box list and raster data preparation. The first three are implemented in the prototype. These tasks were implemented within the object model. The implementation of the data preparation process is illustrated in Figure 4.5.

Algorithm 1: Algorithm of the general workflow for animation production software.

Input : ASPEC-L code.
Output: Animated web map.

```
1  begin
2      // ASPEC-L file path
3      filePath ← "input/file.aspecl"
4      // Source code analyzer
5      // Lexical and syntactic analysis
6      animationObject ← parserASPECL(filePath)
7      if animationObject = Null then
8          Report error
9          Finalize system
10     end
11     // Semantic analysis
12     if animationObject.validateObject() = False then
13         Report error
14         Finalize system
15     end
16     // Animation producer
17     // Data preparation
18     if animationObject.prepareData() = False then
19         Report error
20         Finalize system
21     end
22     // Animation generation
23     if produceAnimation(animationObject) = False then
24         Report error
25         Finalize system
26     end
27 end
```

Algorithm 2: Algorithm for lexical and syntactic analysis. DFA implementation logic.

Input : ASPEC-L code.
Output: Object model.

```
1  begin
2      while not end of file do
3          keyword, value ← getTokens()
4          switch STATE do
5              // Other states conditions
6              ...
7              case "INIT"
8                  if keyword = TOKEN_ANIMATION then
9                      STATE ← "ANIMATION"
10                     animationObject ← animationClass.create()
11                 end
12                 else
13                     STATE ← "FAIL"
14                     Break while loop
15                 end
16             end
17             case "ANIMATION"
18                 if keyword = TOKEN_LAYER then
19                     STATE ← "LAYER"
20                     layerObject ← layerClass.create()
21                 end
22                 else if keyword = TOKEN_END then
                   STATE ← "OBJECTMODEL"
23                 else if keyword is valid property then
24                     animationObject.addProperty(keyword, value)
25                 end
26                 else
27                     STATE ← "FAIL"
28                     Break while loop
29                 end
30             end
31             // Other states conditions
32             ...
33         endsw
34     end
35     if STATE = "FAIL" then
36         Report error
37         animationObject ← Null
38     end
39     RETURN(animationObject)
40 end
```

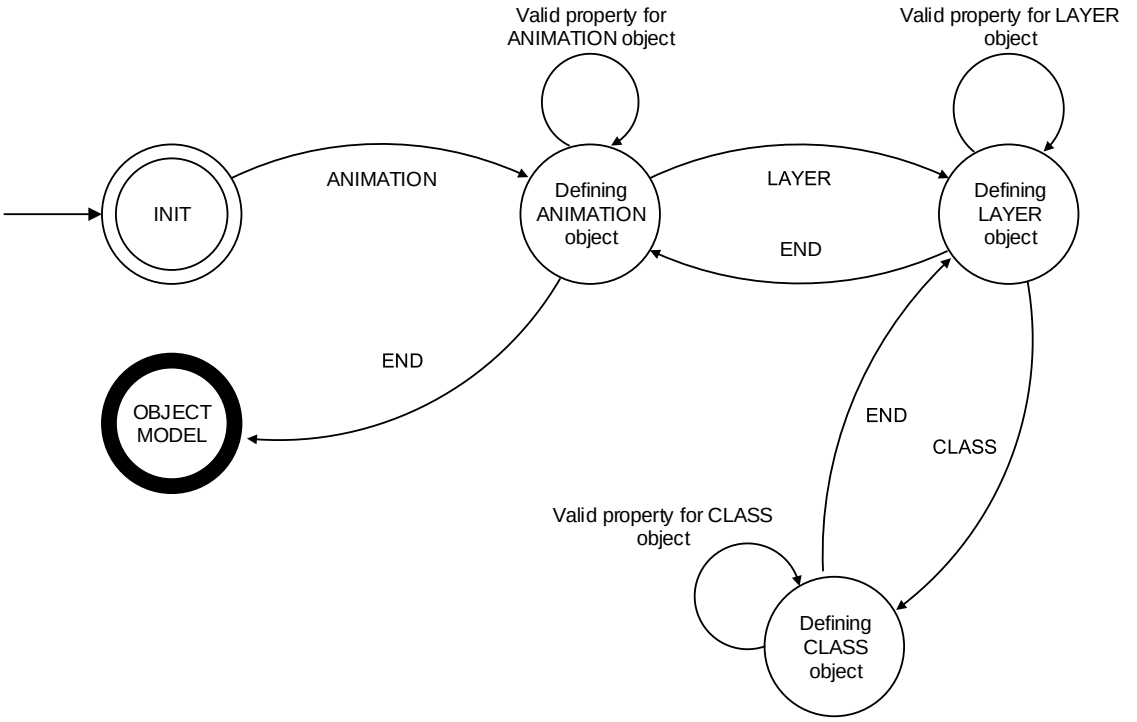


Figure 4.4: DFA design for the parserASPECL function.

The vector data preparation was built using the OGR and Pandas libraries. The OGR library is used to load the data into a local table. This table is built using the *DataFrame* object of Pandas library. The data loading procedure is data source-dependent. Therefore, a specific function is required to retrieve data for each type of data source. In the workflow of vector data preparation, a single call is performed to a generic load data procedure, which selects the appropriate load data function based on the specified data source. Once the data is loaded, the rest of the procedure is generic as indicated in Chapter 3, and is performed over the local table.

The temporal scale factor calculation is implemented using the *datetime* library of Python. This procedure can be divided into three simple steps: create two *datetime* objects to represent the start and end time of the animation, subtract the start time from the end time, and divide the difference over the animation duration.

The bounding box list preparation stage produces a series of timestamps and spatial extension values, as shown in Listing 4.1. The process of preparation depends on the value of the ***BBOX_BEHAVIOR*** attribute. If the value is set to ***FIXED*** or ***USER***, the list only contains the start time of the animation with the bounding box specified in the ***BBOX*** attribute. In the other hand, if ***BBOX_BEHAVIOR*** is set to ***AUTO***, the list is prepared following Algorithm 3, which works as follows: the data is divided into regular time segments and for each time segment a bounding box is computed,[1] this ensures that at any moment the current bounding box covers the area in which the phenomenon is taking place. Nevertheless, this produces unnecessary bounding box

[1]When a bounding box is computed, it is modified to fit with the aspect ratio defined by the width and height of the rendering area.

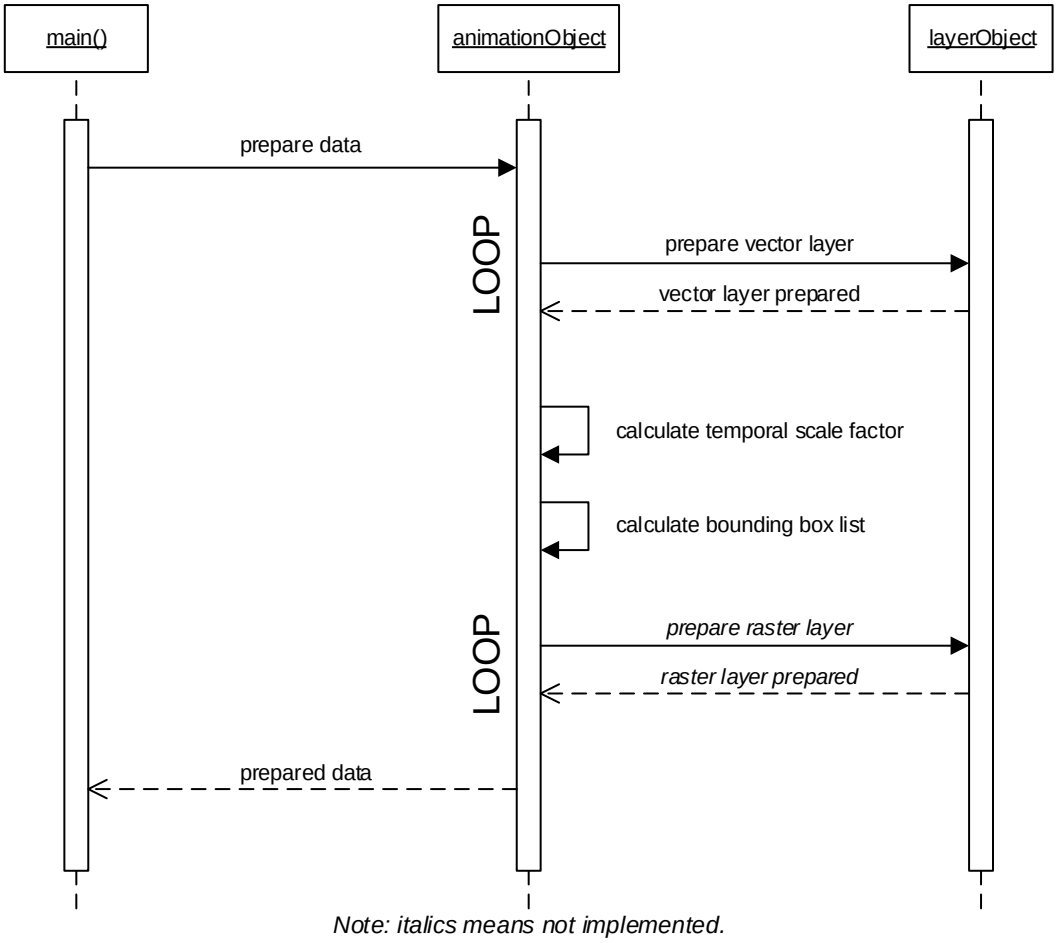


Figure 4.5: Sequence diagram for data preparation.

changes, caused by small differences between previous and current bounding box. Therefore, we implemented a filtering criteria, a bounding box is included in the list if it is completely within the previous one and its area is smaller than a fixed percentage of the area of that bounding box, or if it is not completely within the previous bounding box. When a bounding box is included in the list, it is expanded by a fixed factor, this gives space for the phenomenon to spatially extend until certain point before requiring a new change of bounding box. This filtering criteria stabilizes the screen movement. Once the list is ready, all the bounding boxes on it are expanded to prevent from drawing features too close to the edge of the screen. Finally, linear interpolation is used to smooth the transition between bounding boxes.

Listing 4.1: Bounding box list

```
1 bboxList = [['1996-07-24T00:00:00', -126.98049999999999, 31.431500000000003,
      -88.84450000000001, 60.033500000000004], ['1996-07-24T06:48:08.450000',
      -127.07980766666665, 31.272975500000005, -88.69088233333335, 60.0646695], ... ,
     ['1997-05-31T11:46:46.550000', -127.211409, 32.312182, -88.890161, 61.053118],
     ['1997-05-31T18:34:55', -127.0458, 32.5304, -89.0522, 61.0256]];
```

Algorithm 3: Algorithm to prepare bounding box list for auto-adjustment mode.

Input : Layer to use for auto-adjustment bounding box.
Output: List of bounding boxes.

```
1  begin
2      // Variables initialization
3      startTime ← animationObject.attributes[START_TIME]
4      endTime ← animationObject.attributes[END_TIME]
5      // 5 seconds in animation time
6      timeStep ← animationObject.attributes[TIME_SCALE] * 5
7      bboxList ← Null
8      previousBbox ← Null
9      while startTime < endTime do
10         tempLayer ← layer.getDataBetween(startTime, startTime + timeStep)
11         if tempLayer is not empty then
12             bbox ← tempLayer.getExtent()
13             if includeBbox(previousBbox,bbox) = True then
14                 bbox ← expandBbox(bbox)
15                 previousBbox ← bbox
16                 bboxList.addBbox(bbox)
17             end
18         end
19         startTime ← startTime + timeStep
20     end
21     bboxList ← expandBbox(bboxList)
22     bboxList ← smoothBboxTransition(bboxList)
23 end
```

To illustrate the bounding box list preparation, an example is provided in Figure 4.6. In the example, the bounding box for *time segment 1* is expanded and included in the list (Figure 4.6a). The bounding box for *time segment 2* is computed, but as it is completely within the previous one and its area is not smaller enough to be included, it is discarded (Figure 4.6b). Finally, the bounding box for *time segment 3* is computed and as it is not completely within the previous one, it is expanded and included in the list (Figure 4.6c). To complete the preparation, all the bounding boxes in the list are expanded to ensure that no features will be drawn too close to the edges of the animation (Figure 4.6d). And to smooth the transition between bounding boxes, intermediate ones are computed using linear interpolation (Figure 4.6e). The implemented algorithm is too simple and doesn't fit properly to all the datasets. A better algorithm to prepare the bounding box list, should not rely in fixed factors, but dynamically adjust them to better represent the behavior of the data.

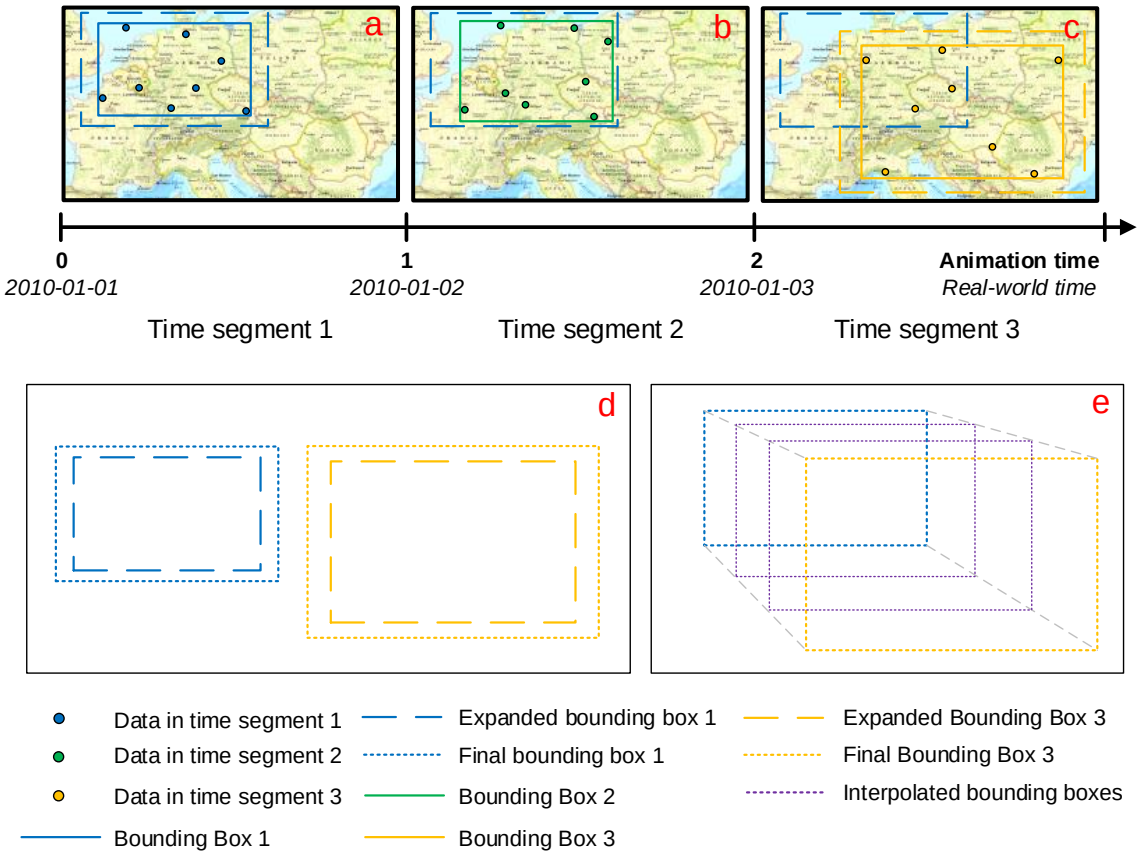


Figure 4.6: Procedure to compute bounding box list.

The implementation of the raster data preparation present several technical challenges. To design the algorithm to prepare the images, it is necessary to be aware that we must prepare as few images as possible and they should be as light as possible, but with enough resolution to properly represent the phenomena in the animation. This is important because a large number of images or heavy images will lead to poor performance of the playback, and bad quality images can mislead the user in the analysis of the phenomena. Regarding to the data, the algorithm must take into account: the dimensions of the images, spatial and radiometric resolution, number of bands, and

the presence of pyramids. Additionally, the spatial extension covered by the images is defined by the bounding box list. Nevertheless, it doesn't mean that we should prepare one image for each bounding box in the list, but that the algorithm should be capable to prepare images that are valid for several bounding boxes. The usage of one image for several bounding boxes is constrained by the scale changes caused by the zooming movement.

4.2.3 Animation production

As indicated in Chapter 3, the animation production is format-dependent. By now, the prototype can produce the animation in WebGL. Our implementation for this output format can be divided in two steps: first, produce the vector animation code, and second, merge the code with the rendering component. More details of the rendering component are provided in Section 4.3. The final output is a web application ready to be disseminated.

The code for vector animation is produced in CZML. "CZML is a subset of JSON, meaning that a valid CZML document is also a valid JSON document. Specifically, a CZML document contains a single JSON array where each object-literal element in the array is a CZML packet. A CZML packet describes the graphical properties for a single object in the scene, such as a single aircraft." (AGI, 2013a). An example of CZML is provided in Listing 4.2. The example shows an animation with interpolation for position, color and size. In the example, the reader can notice a special packet which *id* property value is *document*, this packet contains the start and end time of the animation, and the temporal scale factor. The full description of CZML can be found in the official page of the project.[2]

Listing 4.2: CZML example

```
[
   {
    "id":"document",
    "clock":{
      "interval":"2004-09-02T18:00:00/2004-09-24T06:00:00",
      "currentTime":"2004-09-02T18:00:00",
      "multiplier":61920.0,
      "range":"LOOP_STOP",
      "step":"SYSTEM_CLOCK_MULTIPLIER"
     }
   },
   {
    "id": "IVAN",
    "availability": "2004-09-02T18:00:00/2004-09-24T06:00:00",
    "point": {
      "color": {
        "interpolationAlgorithm":"LAGRANGE",
        "interpolationDegree":1,
        "epoch":"2004-09-02T18:00:00",
        "rgba": [
          "2004-09-02T18:00:00",0,255,0,255,
          ...
          "2004-09-24T06:00:00",0,255,0,255
```

[2] https://github.com/AnalyticalGraphicsInc/cesium/wiki/CZML-Guide

```
        ]
      },
      "outlineColor": {
        "rgba": [
          0,0,0,255
        ]
      },
      "outlineWidth": {
        "number": 2
      },
      "pixelSize":{
        "interpolationAlgorithm":"LAGRANGE",
        "interpolationDegree":1,
        "epoch":"2004-09-02T18:00:00",
        "number": [
          "2004-09-02T18:00:00",12.5,
          ...
          "2004-09-24T06:00:00",12.5
        ]
      }
    },
    "position": {
      "interpolationAlgorithm": "LAGRANGE",
      "interpolationDegree":1,
      "epoch": "2004-09-02T18:00:00",
      "cartographicDegrees": [
        "2004-09-02T18:00:00",-27.6,9.7,0,
        ...
        "2004-09-24T06:00:00",-94.2,30.1,0
      ]
    }
  }
]
```

Depending on the type of animation to be depicted, the CZML packets are created with different setups. To create existential animation, the *availability* attribute of the packet is used to indicate the lifetime of the object, and no interpolation is indicated in position or any of the visual properties. To depict stepwise animation, for each object several packets are created, one for each state of it. This means that each packet is an existential change that depicts one state of the object. Finally, to represent interpolated animation with point data, one packet is created for each object, the interpolated properties are specified within it. In Appendix C, examples of CZML code for the three types of animation are provided. The examples include existential, stepwise and interpolated animation with point data; and, existential and stepwise with line and polygon data. Finally, we couldn't find in CESIUM mechanisms to produce interpolated animation with lines and polygons. If required, this functionality can be built directly with WebGL, but as WebGL is a very low-level library, we foreseen that this development will require a significant coding effort. Algorithms that can be implemented for interpolation in polygons and images are described in (Málková et al., 2010) and (Mahajan et al., 2009) respectively.

4.3 ANIMATION RENDERER

The rendering component was built as a web application. This web application uses the *Dynamic scene* layer of the CESIUM library to render CZML files. To use the *Dynamic scene*, the *viewer* object is instantiated and the CZML file is loaded into it. This object includes controls for play, pause, panning, zooming, temporal legend, playing direction, change of base map and speed control. Figure 4.7 shows the design of the interface. Two functions are important in this web application: init function, which provides a set-up of the initial state of the web application, and the tick function, which updates the spatial extension. Finally, as indicated before, the prototype is not capable to produce raster-based animation, but the rendering component is already capable to play it. An example of this is shown in Figure 4.8.

4.4 SUMMARY

In this chapter, we described the development of a prototype that implements our approach for animated web map production. This prototype implements just a small part of the design presented in Chapter 3. Nevertheless, it is useful to demonstrate the implementation of the approach in a functional system. Even when the prototype is only capable at present of consuming data from WFSes, it is not constrained to a dataset with a tailor-made structure and the implementation of new vector data sources should not represent a challenge. The only output format implemented is WebGL. The implementation of new output formats will require a considerable development effort, but the usage of these is transparent for the final user, who only needs to change the specified value for ***FORMAT*** attribute in ***ANIMATION*** object.

Figure 4.7: Web application interface.

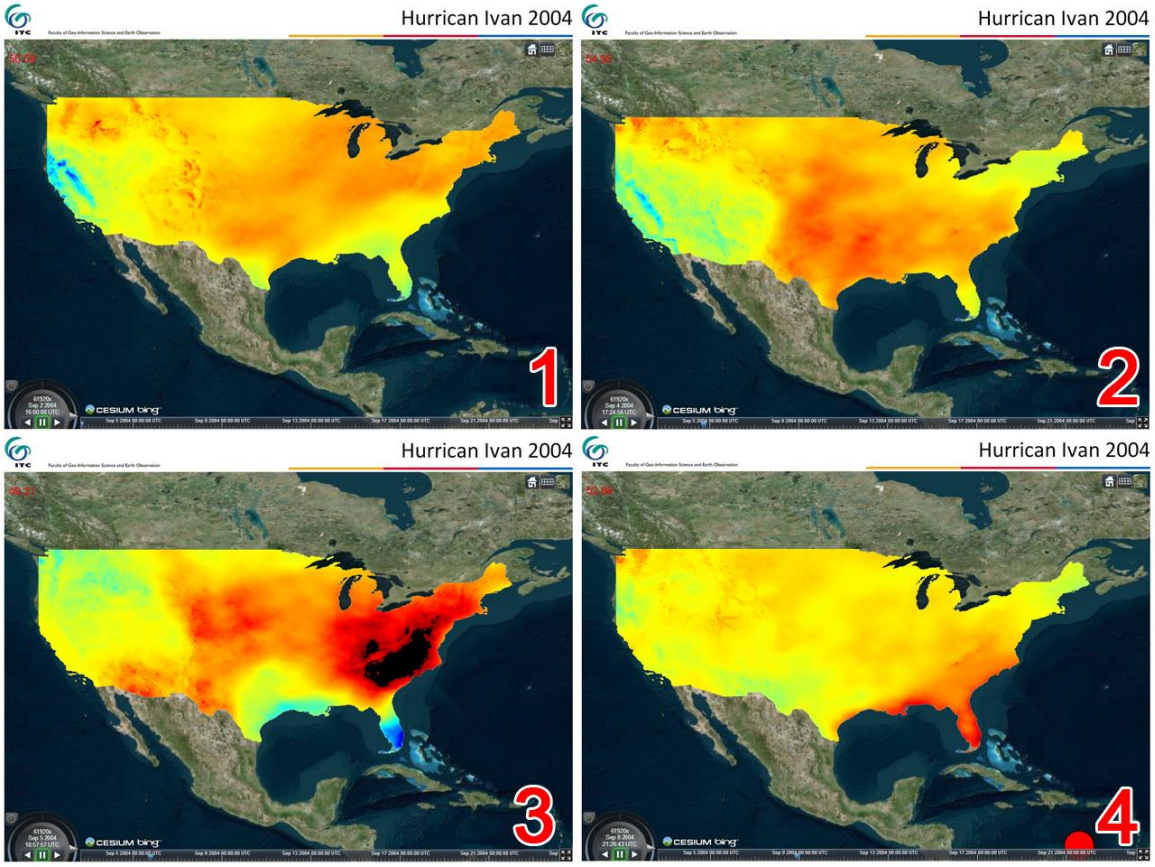


Figure 4.8: Animation renderer - raster-based animation test.

Chapter 5
The toolset in action: application cases

In this chapter, we are presenting the results of using the prototype to build animations for three application cases: Lilac blooming in USA, Swainson's Hawk migration in the Americas, and the movement of hurricanes on the Pacific Ocean. These cases were used for two purposes: first, to check if the features of the language and toolset are implemented properly in the prototype; and second, to measure the performance of the toolset in producing the animations, and the performance of the rendering component on playing the animations.

The performance test for animation production was done by running the toolset five times for each specification. In total 17 animations were built.[1] Therefore the toolset was executed 85 times. The measurement used was the number of seconds required to produce the animation.

The performance test for animation playback was done by playing the animations five times. The measurement used in this test was the number of Frames Per Second (FPS). Each measurement is the average over hundred frames, and for each animation twenty measurements were taken. For this test, we wanted to use Internet Explorer, Mozilla Firefox and Google Chrome. However, Internet Explorer couldn't be used, because it crashes when loading the CESIUM library.

For each application case, we are presenting the measurements taken, the basic statistics and a small multiple map created from one of animations per case. The ASPEC-L code for all the animations presented in this chapter are included in Appendix B.

5.1 LILAC BLOOMING

Existential animations were created to represent Lilac blooming phenomenon. The dataset used is the North American First Leaf and First Bloom Lilac Phenology Data (Schwartz & Caprio, 2003). This dataset contains observations over the United States for the period from 1956 to 2003. In total it contains 23,338 observations, 9,073 for first leaf and 14,265 for first bloom. This application case was used to analyze the behavior of the toolset with increasing dataset sizes, and the performance of the rendering component with existential animation. Figure 5.1 shows the small multiple map for lilac first leaf and bloom in 1968.

The animations built for this application case are:

1. First leaf and bloom in 1968. This animation includes 1,189 objects.

2. First leaf and bloom from 1970 to 2003. The animation contains 14,186 objects.

[1] Six animations for Lilac blooming, six for Hawk migration and five for Hurricane movement.

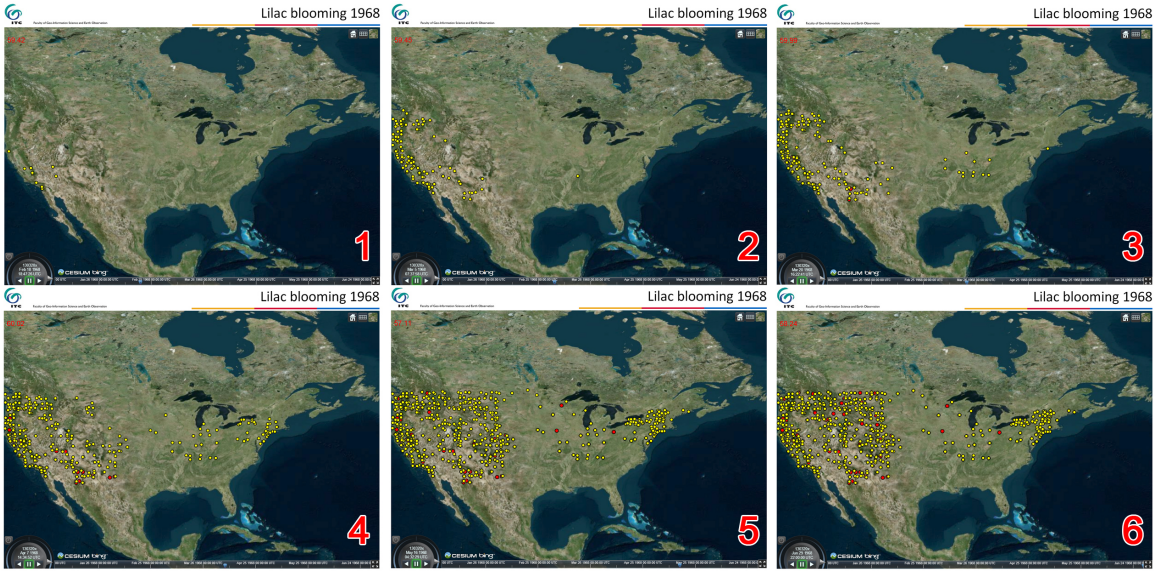


Figure 5.1: Small multiple map of animation for Lilac blooming in 1968.

3. First leaf and bloom from 1956 to 2003. It is the whole dataset, which includes 23,338 objects.
4. Simulated dataset including 2 times the whole data (46,676 objects).
5. Simulated dataset including 4 times the whole data (93,352 objects).
6. Simulated dataset including 8 times the whole data (186,704 objects).

The results of the performance test for animation production are shown in Table 5.1, indicate that the time required to produce an animation is directly related to the dataset size. This result was expected. However, it is interesting that for producing an animation including the whole dataset, this is 23,338 objects, the prototype requires less than a minute. Furthermore, the prototype can produce an animation including more that 180,000 objects, in around 4 minutes. The animation production process shows a linear relation between the number of objects and the time required to produce the animation as can be seen in Figure 5.2. It is interesting to perform future animation production performance test with different types of data.

The results of the playback test for this application case shown in Table 5.2, indicate that the performance of the animation is negatively affected by the number of objects included in it. The effect can be described as exponential for Google Chrome and logarithmic for Mozilla Firefox (See Figure 5.3). In general, Google Chrome performed by far better than Mozilla Firefox. Finally, in the animation with 186,704 objects, delays in the playback and response of the controls were evident.

5.2 SWAINSON'S HAWK MIGRATION

Interpolated animations were created to represent the Swainson's Hawk migration phenomenon. The dataset used is from the study of Swainson's Hawks (*Buteo swainsoni*) (Fuller et al., 1998). This dataset contains 4,514 observations, that describe migration movement of 43 hawks, between July

Table 5.1 Production performance test for Lilac blooming.

	Animation specification *Number of objects included in the animation*					
	1	**2**	**3**	**4**	**5**	**6**
Run	*1,189*	*14,186*	*23,338*	*46,676*	*93,352*	*186,704*
1	11.91	23.36	41.66	57.47	118.94	242.32
2	11.73	23.21	37.84	55.66	110.20	233.00
3	11.37	22.57	36.88	57.33	119.06	219.21
4	12.46	24.94	34.7	53.59	130.40	229.93
5	11.35	26.24	37.48	52.88	135.53	236.80
Minimum	11.35	22.57	34.7	52.88	110.2	219.21
Maximum	12.46	26.24	41.66	57.47	135.53	242.32
Mean	11.76	24.06	37.71	55.39	122.83	232.25
Std. dev.	0.41	1.34	2.25	1.88	9.03	7.72

Note: all measurements are in seconds.

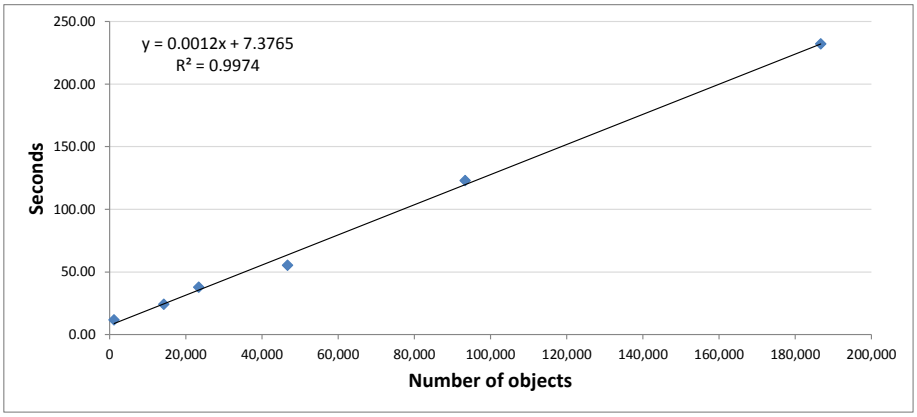
1995 and June 1998. The phenomenon extends over the Americas. This application case was used to analyze the performance of the toolset with different specifications for visual representation and spatial extension behavior. Figure 5.4 shows the small multiple map for hawks migration movement between July 1996 and June 1997, using four symbolization classes and auto-adjustment spatial extension.

To avoid the effect of dataset size, the three animations for this application case were built using a subset that covers the period from July 1996 to June 1997, it includes 3,318 objects, which represent the movement of 28 hawks. The specifications to build the animations are:

1. Same symbolization for all the objects and fixed extension.
2. Two symbolization classes and fixed extension.
3. Four symbolization classes and fixed extension.
4. Eight symbolization classes and fixed extension.
5. Sixteen symbolization classes and fixed extension.
6. One symbolization class and auto-adjustment extension.

The results shown in Table 5.3 indicate that as effect of the extra computations required when using classes and auto-adjustment of spatial extent, the animation production time increases. In the case of the number of classes, the results shown in Figure 5.5, indicates that the increase of production time can be explain as a linear behavior. Even when the quadratic approximation shows a lightly better R^2 value, the difference is not significant. However, for a better assessment of the effects of using classes and auto-adjustment spatial extent, further tests are required using different applications cases.

The playback test for this application case shown a performance above 45 frames per second in every case (Table 5.4). This result is interesting given that the rendering component has to adjust the spatial extent and to interpolate positions on the fly, while playing the animation. The number of classes seems to have no impact in the playback performance, given that the animations using one, two, four, eight and sixteen classes shown the same performance. In all cases, the interactive



DATA TABLE						
Objects	1,189	14,186	23,338	46,676	93,352	186,704
Seconds measured	11.76	24.06	37.71	55.39	122.83	232.25
Seconds estimated	8.80	24.40	35.38	63.38	119.40	231.42
Difference	2.96	-0.33	2.33	-7.99	3.43	0.83

Figure 5.2: Behavior of the toolset in animation production with increasing dataset sizes.

controls worked without any problem.

5.3 HURRICANE MOVEMENT

Stepwise and interpolated animations were created to represent hurricane movement phenomena. The dataset used was obtained from UNISYS (2014). The dataset contains observations over the Pacific Ocean for the period between 1950 and 2012. In total it contains 25,308 observations, which represent the movement of 941 hurricanes. This application case was used to compare the performance between stepwise and interpolated animations. Figure 5.6 shows the small multiple map for Hurricane Ivan in 2004, using interpolation in position, size and color.

The animations built for this case shown the movement of Hurricane Ivan in 2004. There are 94 records that belong to this hurricane in the dataset. The specifications to build the animations are:

1. Stepwise animation.
2. Animation with interpolation for location.
3. Animation with interpolation for location and size.
4. Animation with interpolation for location and color.
5. Animation with interpolation for location, size and color.

The results of the animation production test for this application case (Table 5.5), show almost no difference in producing stepwise and interpolated animation. It is important to consider that the interpolation is not performed during the animation production, but during animation playback.

Table 5.2 Playback performance test for Lilac blooming.

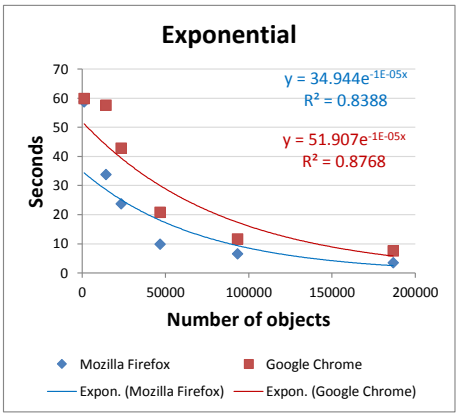
		Specification *Number of objects included in the animation*					
Mozilla Firefox		**1**	**2**	**3**	**4**	**5**	**6**
	Run	*1,189*	*14,186*	*23,338*	*46,676*	*93,352*	*186,704*
	1	58.55	36.11	25.17	9.47	9.32	3.35
	2	58.10	32.12	22.71	9.70	5.32	5.45
	3	58.88	35.71	23.34	9.80	6.18	2.06
	4	58.79	33.44	24.82	9.81	6.06	3.16
	5	59.11	31.83	22.91	10.31	5.82	3.35
	Minimum	58.10	31.83	22.71	9.47	5.32	2.06
	Maximum	59.11	36.11	25.17	10.31	9.32	5.45
	Mean	58.69	33.842	23.79	9.81	6.54	3.47
	Std. dev.	0.34	1.78	1.01	0.27	1.42	1.10
Google Chrome		**1**	**2**	**3**	**4**	**5**	**6**
	Run	*1,189*	*14,186*	*23,338*	*46,676*	*93,352*	*186,704*
	1	59.82	59.20	42.31	21.73	11.73	8.27
	2	59.86	59.22	46.76	21.79	11.34	6.89
	3	59.88	58.35	42.93	21.30	10.79	8.50
	4	59.91	55.03	40.09	19.69	11.91	6.43
	5	59.90	56.26	41.89	19.37	12.39	7.61
	Minimum	59.82	55.03	40.09	19.37	10.79	6.43
	Maximum	59.91	59.22	46.76	21.79	12.39	8.5
	Mean	59.87	57.61	42.80	20.78	11.63	7.54
	Std. dev.	0.03	1.68	2.19	1.04	0.54	0.79

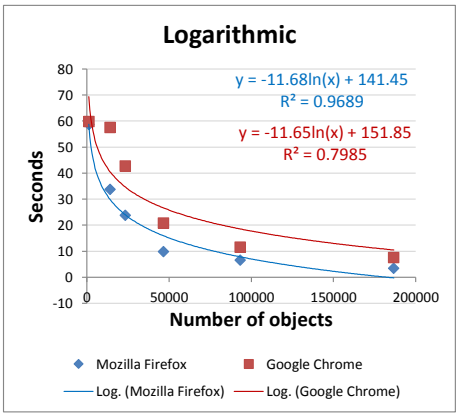
Note: all measurements are in Frames Per Second (FPS).

The results shown in Table 5.6 indicate that interpolation negatively affects the performance of the animation playback. This result is not clear when comparing stepwise animation and the animation with position interpolation. But it is evident when compared to the animation with interpolation for position, size and color. Additionally, it seems that the interpolation of color is the most computationally expensive. Contrary to the result obtained in the Lilac blooming case, Google Chrome is outperformed by Mozilla Firefox. However, in all cases the animation is fluid and interactive controls works properly.

5.4 SUMMARY

In this chapter, we presented the results of the prototype's performance test. In general, the prototype has shown good performance to produce and play the animations. Almost all the animations were produced in less than 1 minute, the playback was fluid and the interactive controls respond without delays. However, for producing an animation with around 180,000 objects, the prototype required 4 minutes, and the decrease of playback performance and responsiveness of interactive controls was evident. In general, the performance was mainly affected by the number of objects in the animation. Further testing with different application cases and specifications is required to validate the results presented in this chapter. Additionally, it is important to perform an usability test, which should evaluate the usefulness of the toolset and the produced animations.





DATA TABLE						
Objects	1,189	14,186	23,338	46,676	93,352	186,704
Mozilla Firefox	58.69	33.84	23.79	9.82	6.54	3.47
Google Chrome	59.87	57.61	42.80	20.78	11.63	7.54

Figure 5.3: Behavior of the toolset in animation playback with increasing dataset sizes.

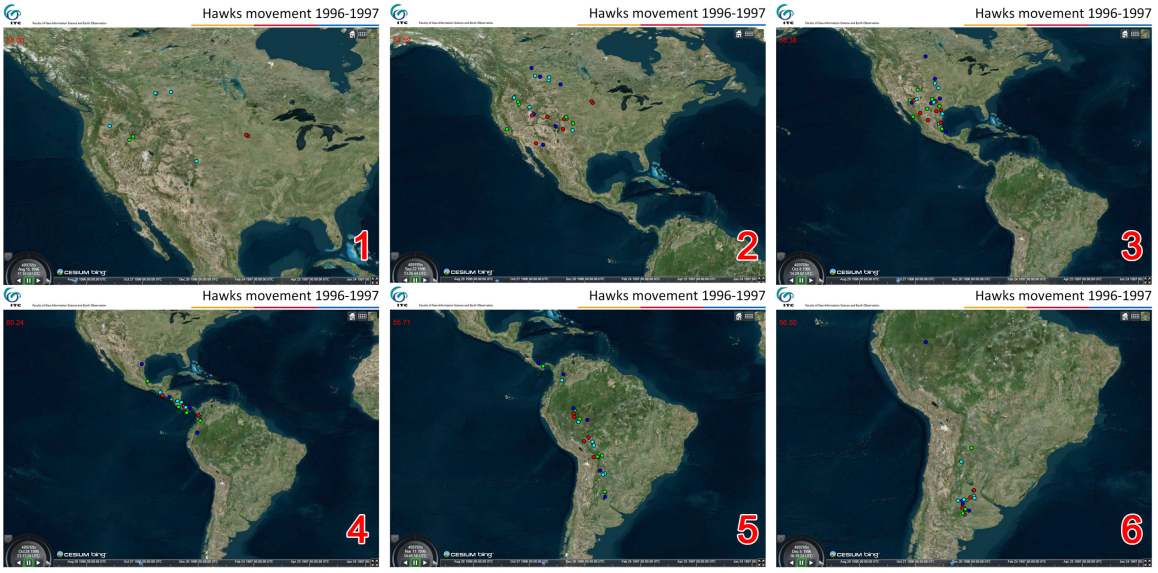


Figure 5.4: Small multiple map of animation for Hawk migration 1996-1997.

Table 5.3 Production performance test for Swainson's Hawk migration.

	Animation specification *Number of classes and bounding box behavior*					
Run	**1** *One class and fixed extent*	**2** *Two classes and fixed extent*	**3** *Four classes and fixed extent*	**4** *Eight classes and fixed extent*	**5** *Sixteen classes and fixed extent*	**6** *One class and auto-adjustment extent*
1	5.18	5.79	6.27	6.42	8.23	7.35
2	5.18	5.11	5.69	6.26	8.21	6.67
3	4.75	5.36	7.04	6.53	8.15	7.2
4	4.82	4.98	6.54	7.21	8.01	8.21
5	55.16	4.95	6.84	6.22	8.76	7.76
Minimum	4.75	4.95	5.69	6.22	8.01	6.67
Maximum	5.18	5.79	7.04	7.21	8.76	8.21
Mean	4.98	5.24	6.48	6.53	8.27	7.44
Std. dev.	0.19	0.31	0.47	0.36	0.26	0.52

Note: all measurements are in seconds.

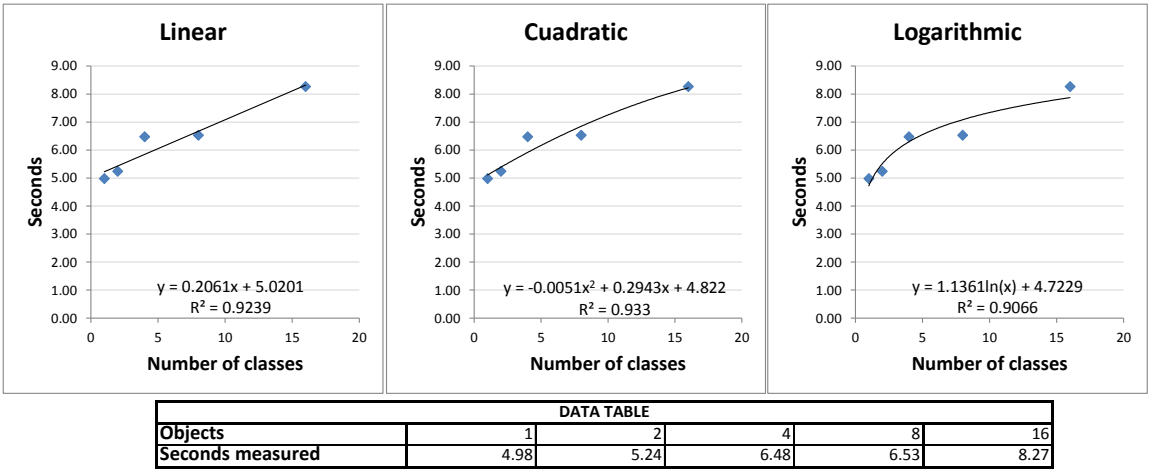


Figure 5.5: Behavior of the toolset in animation production with increasing number of classes.

Table 5.4 Playback performance test for Swainson's Hawk migration.

		Animation specification *Number of classes and bounding box behavior*					
Mozilla Firefox	**Run**	**1** *One class and fixed extent*	**2** *Two classes and fixed extent*	**3** *Four classes and fixed extent*	**4** *Eight classes and fixed extent*	**5** *Sixteen classes and fixed extent*	**6** *One class and auto-adjustment extent*
	1	59.62	59.47	58.77	58.29	58.57	51.11
	2	59.81	59.22	59.76	59.47	59.31	47.52
	3	59.51	58.28	59.71	59.43	59.51	46.04
	4	59.39	57.15	59.60	58.87	58.28	44.54
	5	58.88	55.89	59.72	58.88	56.02	45.35
	Minimum	58.88	55.89	58.77	58.29	56.02	44.54
	Maximum	59.81	59.47	59.76	59.47	59.51	51.11
	Mean	59.44	58.00	59.51	58.99	58.34	46.91
	Std. dev.	0.31	1.33	0.37	0.43	1.24	2.32
Google Chrome	**Run**	**1** *One class and fixed extent*	**2** *Two classes and fixed extent*	**3** *Four classes and fixed extent*	**4** *Eight classes and fixed extent*	**5** *Sixteen classes and fixed extent*	**6** *One class and auto-adjustment extent*
	1	59.76	59.75	59.81	58.94	59.72	51.89
	2	59.84	59.06	59.85	56.12	59.88	51.39
	3	59.85	59.90	59.92	54.36	59.10	50.09
	4	59.84	59.71	59.94	55.17	55.23	51.73
	5	59.86	59.80	59.59	56.16	55.32	49.69
	Minimum	59.76	59.06	59.59	54.36	55.23	49.69
	Maximum	59.86	59.90	59.94	58.94	59.88	51.89
	Mean	59.83	59.64	59.82	56.15	57.85	50.96
	Std. dev.	0.04	0.30	0.13	1.55	2.12	0.90

Note: all measurements are in Frames Per Second (FPS).

Table 5.5 Production performance test for Hurricane movement.

	Animation specification *Interpolated properties*				
Run	**1** *No interpolation*	**2** *Location*	**3** *Location and size*	**4** *Location and color*	**5** *Location, size and color*
1	17.03	18.38	16.68	17.68	18.33
2	17.27	18.83	16.90	17.19	17.80
3	18.38	17.59	20.06	16.06	16.44
4	17.69	16.01	15.80	17.56	15.97
5	15.52	17.66	16.27	17.77	17.88
Minimum	15.52	16.01	15.80	16.06	15.97
Maximum	18.38	18.83	20.06	17.77	18.33
Mean	17.22	17.69	17.14	17.25	17.28
Std. dev.	1.06	0.96	1.51	0.63	0.91

Note: all measurements are in seconds.

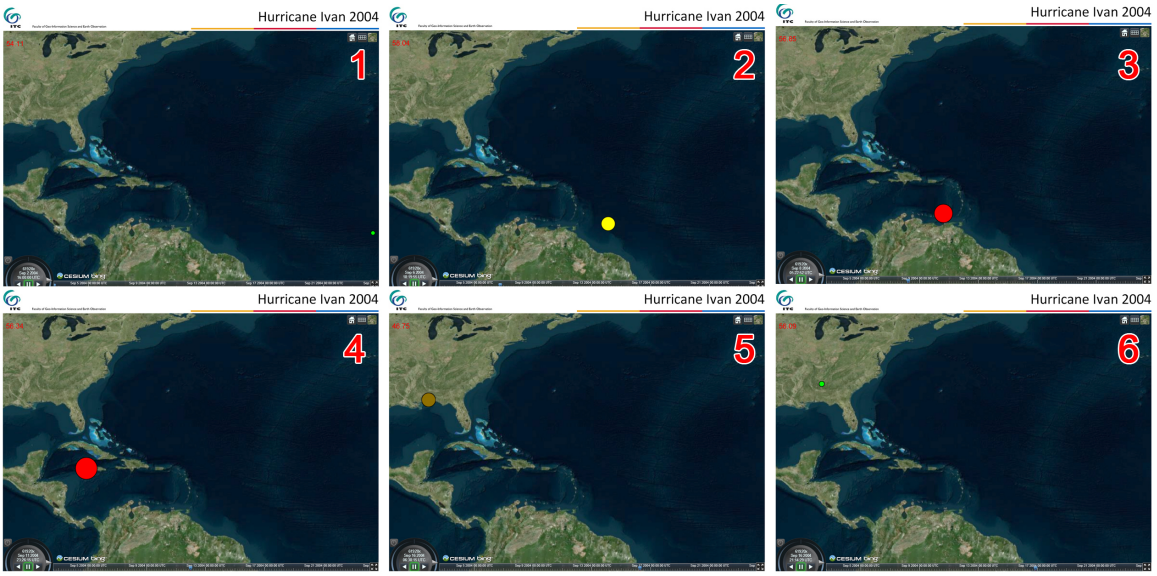


Figure 5.6: Small multiple map of animation for Hurricane Ivan 2004.

Table 5.6 Playback performance test for Hurricane movement.

		Animation specification *Number of classes and bounding box behavior*				
Mozilla Firefox	**Run**	**1** *No interpolation*	**2** *Location*	**3** *Location and size*	**4** *Location and color*	**5** *Location, size and color*
	1	58.23	58.37	56.81	56.14	54.33
	2	58.12	58.5	56.94	54.23	53.33
	3	58.19	58.66	57.511	53.6	52.46
	4	58.42	58.69	57.76	56.35	53.33
	5	58.59	58.14	57.03	52.06	52.17
	Minimum	58.12	58.14	56.81	52.06	52.17
	Maximum	58.59	58.69	57.76	56.35	54.33
	Mean	58.31	58.47	57.21	54.48	53.12
	Std. dev.	0.17	0.20	0.36	1.61	0.76
Google Chrome	**Run**	**1** *No interpolation*	**2** *Location*	**3** *Location and size*	**4** *Location and color*	**5** *Location, size and color*
	1	49.97	48.14	47.56	46.47	44.84
	2	49.7	48.39	45.98	45.21	45.34
	3	49.29	48.59	48.01	44.85	44.68
	4	48.48	48.53	46.89	46.96	44.62
	5	48.12	48.18	47.13	45.73	44.61
	Minimum	48.12	48.14	45.98	44.85	44.61
	Maximum	49.97	48.59	48.01	46.96	45.34
	Mean	49.11	48.37	47.11	45.84	44.82
	Std. dev.	0.71	0.18	0.68	0.78	0.27

Note: all measurements are in Frames Per Second (FPS).

Chapter 6
Discussion, conclusions and recommendations

In this chapter, we discuss relevant aspects of the design, development and testing of our language and toolset for animated web map production. Based on this discussion, we draw our conclusions and recommendations for further research.

6.1 DISCUSSION ON OUR APPROACH FOR ANIMATED WEB MAP PRODUCTION

In the present research project, we designed an approach for the production of animated web map based on a declarative language. The Animation SPECification Language (ASPEC-L) aims to specify the basic characteristics of spatio-temporal animation in raster and vector domain. The basic characteristics include: the type of change to be represented, the data to be used and the visual representation of the objects in the animation. By the end of this project, ASPEC-L consists of 91 lexical elements, and a syntax based on a hierarchical structure inspired by MapServer Mapfile syntax. This syntax was chosen because of its simplicity and the possibility of extending the language definition. The language definition can be extended by adding new attributes to the existing objects and by defining new objects.

We designed a toolset capable of producing animations from the specifications written in ASPEC-L. Given that the mechanism to specify the animated maps is a language, we based the design of the toolset on a generic compiler. This decision led us to a basic architecture with two components: source code analyzer and animation producer. Further analysis of the architecture showed the necessity of adding a third component, the animation renderer. The final architecture is shown in Figure 3.4. The process of animation production is performed by the first two components, which execute three processes: source code analysis, data preparation and animation production. The communication between and within these components is performed using a data structure that forms an object model. The third component is a mechanism to play the animation on a web browser, it can be either a web application or a plug-in. The separation of the components as indicated above, provides a loosely coupled architecture, which facilitates the maintenance, substitution and extension of the toolset components.

As proof of concept, we developed a prototype which implements our approach for animated web map production. The development was done following an incremental approach. To guide the development, a five-step plan was prepared, from it, the first four steps were accomplished (see Table 4.1). The plan was designed with the ambition of producing animations including two layers, in the background a raster-based animation providing spatial context (e.g., temperature) for the main phenomena presented in the foreground layer as a vector-based animation (e.g., bloom of flowers). We think that this combination could be useful when the main phenomena is affected or driven by an underlying phenomena.

On its current level of development, the prototype is capable of producing animations to represent existential, spatial and attribute changes. The spatial and attribute changes can be represented in stepwise or interpolated mode. The existential and stepwise animation can be produce using point, line and polygon data, and the interpolated animations can be produced using point data. Regarding the input and output formats, the prototype is capable of consuming data from WFSes and producing the animation in WebGL. Even when only one data source type is supported, the prototype is not constrained to a dataset with a tailor-made structure and the implementation of new vector formats should not represent a challenge. On the other hand, the implementation of new output formats require a considerable coding effort.

By now the prototype doesn't implement the production of raster-based animation. By analyzing the characteristics of raster data, we determined that the implementation of this feature will require a significant designing and coding effort. This is due to all the factors that need to be taken into account, among them: number of images to be produce, size and resolution of those images, spatial extents determine by the bounding box list and characteristics of the data itself (dimensions of the images, spatial and radiometric resolution, number of bands, and the presence of pyramids). It is necessary to optimize this procedure, to minimize the negative impact on the animation playback performance.

The use of the prototype to produce animated web maps for three application cases, showed that ASPEC-L provides a flexible mechanism to specify animated maps. Additionally, these application cases were used to test the performance of the animation production and playback. In both tests, the prototype performed well. We determined that the performance in animation production and playback is mainly affected by the dataset size.

By now, it is only possible to specify one animation per ASPEC-L file. but while testing the prototype with the application cases, we observed that once a specification is written, it can be reused to produce different animations for the same phenomena requiring minimum changes. For example, in the hurricane movement application case, the production of animations for different hurricanes only requires to change the filtering criteria and spatial extent. This characteristic of our approach clearly reduces the required time to produce series of animated maps. This characteristic can be further improved by extending the language definition to allow series of values for attributes. This means the possibility of specify series of animations in a single ASPEC-L file.

As discussed earlier, the design was intentionally restricted to the basic characteristics of spatio-temporal animation. From the analysis of the produced animation for the application cases, we think that they can be further improved by adding characteristics such as: paths describing the movement of the objects, convex hulls surrounding groups of related objects, timestamped annotations and images, and parallel views. These features can be helpful in the analysis and presentation of dynamic phenomena.

The toolset was designed as a standalone application, which reads an ASPEC-L file and produces an animated map that can be disseminated through a web server. This architecture is functional, but we found that on the iterative process of designing an animation, a lot of time is spent in copying files to the web server. Therefore, moving from the actual design of the toolset, to one allowing the user to do dynamic coding can improve the usability of this approach. By dynamic coding, we mean that the system will provide an interface based on a web application that allows the user to code and test the animation directly on the web browser. An example of such an archi-

tecture is shown in Figure 6.1.

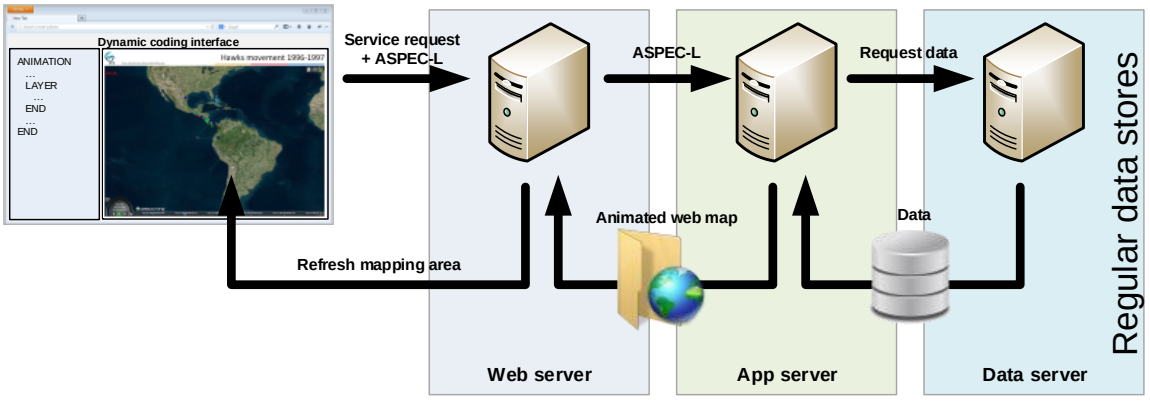


Figure 6.1: High-level architecture of a toolset with support for dynamic coding.

Finally, we found interesting the idea of combine the work done by Bekele (2014) and Gudecha (2014), with our work. Bekele (2014) worked in the extraction of trajectories from historical texts and Gudecha (2014) worked on the development of a data type to provide support for moving objects in a DBMS. The combination of this three projects can lead to the design of a tool for the reconstruction and analysis of historical events. Such a tool can support the acquisition, storage, analysis, visualization and communication of spatio-temporal data from historical logbooks.

6.2 CONCLUSIONS AND RECOMMENDATIONS

The design of the ASPEC-L demonstrates the possibility of specifying spatio-temporal animation by means of a declarative language. We suggest further research to assess the learning curve of the language and to evaluate the possibilities to extend the language specification.

On providing a mechanism to produce animations from the specification written in ASPEC-L, we succeeded in the design of a toolset for this aim. The architecture of the toolset includes three components: source code analyzer, animation producer and animation renderer. This separation of the components, and the communication mechanism between them, facilitate the maintenance and extension of the toolset. To improve the usability of our approach, we propose to modify the toolset architecture to allow dynamic coding.

The development of the prototype proved that our approach for animated web maps can be implemented in a functional system. The implemented features demonstrate that the language and the toolset work according to the design. To improve the results of this research, we recommend the development of a complete system, which can be used to assess the full potential of our approach.

The application cases showed that ASPEC-L provides a mechanism to specify spatio-temporal animation, that can be useful in different fields of knowledge. Additionally, the application cases showed that the system performs well in the production and playback of the animations. By testing

different settings for each application case, we determined that the performance is mainly affected by the dataset size. To complete the evaluation of the system, it is required to perform an usability test, which will help to determine if the system is easy to use and if the produced animations are useful to analyze the phenomena under study.

Finally, we can conclude that the designed approach provides a systematic and consistent procedure for the production of animated web maps, that has the potential to overcome the identified deficiencies (required time for animated map production, use of regular data sources and performance issues) in the currently available systems.

References

Adobe. (2014). *Flash glossary: Shape tween.* Retrieved 2014-03-05, from `http://www.adobe.com/devnet/flash/articles/concept_shape_tween.html`

AGI. (2013a). *Cesium: Webgl virtual globe and map engine.* Retrieved 2013-10-05, from `http://www.cesiumjs.org/`

AGI. (2013b). *Czml guide.* Retrieved 2013-10-05, from `https://github.com/AnalyticalGraphicsInc/cesium/wiki/CZML-Guide`

Andrienko, G., Andrienko, N., Demsar, U., Dransch, D., Dykes, J., Fabrikant, S. I., … Tominski, C. (2010). Space, time and visual analytics. *International Journal of Geographical Information Science*, *24*(10), 1577-1600.

Andrienko, N., Andrienko, G., & Gatalsky, P. (2003). Exploratory spatio-temporal visualization: an analytical review. *Journal of Visual Languages & Computing*, *14*, 503-541.

Animaps. (2011). *What are animaps?* Retrieved 2013-08-09, from `http://www.animaps.com/#!home`

Becker, T. (2009). *Visualizing time series data using web map service time dimension and svg interactive animation* (MSc thesis). International Institute for Geo-information Science and Earth Observation (ITC).

Becker, T., Köbben, B., & Blok, C. (2009). Timemapper : visualizing moving object data using wms time and svg smil interactive animations. *In: Proceedings SVGOpen 2009 : 7th international conference on scalable vector graphics, 2-4 October 2009, Mountain View, USA. 13 p.*.

Bekele, M. K. (2014). *Spatial tracing of historic expeditions: from text to trajectory* (Unpublished master's thesis). International Institute for Geo-information Science and Earth Observation (ITC).

Blok, C. (2000). Monitoring change : characteristics of dynamic geo - spatial phenomena for visual exploration. *In: Spatial Cognition II : an interdisciplinary approach to representing and processing spatial knowledge / Ch. Freksa ... [et al.] (eds.) Berlin : Springer verlag, 2000. ISBN 3-540-64603-5 (Lecture Notes in Artificial Intelligence : 1404) pp. 16-30*.

Blok, C. (2005). Dynamic visualization variables in animation to support monitoring. *In: ICC 2005 : Proceedings of the 22nd international cartographic conference : mapping approaches into a changing world, 9-16 July 2005, A CoruÃśa, Spain. International Cartographic Association (ICA), 2005. ISBN: 0-958-46093-0. pp. 10*.

Bostock, M., & Heer, J. (2009). Protovis: A graphical toolkit for visualization. *Visualization and Computer Graphics, IEEE Transactions on*, *15*(6), 1121-1128.

Bostock, M., Ogievetsky, V., & Heer, J. (2011). D3: Data-driven documents. *IEEE Trans. Visualization & Comp. Graphics (Proc. InfoVis).* Retrieved from `http://vis.stanford.edu/papers/d3`

Cinnamon, J., Rinner, C., Cusimano, M. D., Marshall, S., Bekele, T., Hernandez, T., … Chipman, M. L. (2009, NOV). Evaluating web-based static, animated and interactive maps for injury prevention [Article]. *GEOSPATIAL HEALTH*, *4*(1), 3-16.

De Ambros, M., Lanza, M., Lungu, M., & Robbes, R. (2011, April). On porting software visualization tools to the web. *Int. J. Softw. Tools Technol. Transf.*, *13*(2), 181-200. Retrieved from

http://dx.doi.org/10.1007/s10009-010-0171-9 doi: 10.1007/s10009-010-0171-9

Dibiase, D., Maceachren, A. M., Krygier, J. B., & Reeves, C. (1992). Animation and the role of map design in scientific visualization. *Cartography and Geographic Information Systems*, *19*.

Dong, W., Ran, J., & Wang, J. (2012, APR). Effectiveness and efficiency of map symbols for dynamic geographic information visualization [Article]. *CARTOGRAPHY AND GEOGRAPHIC INFORMATION SCIENCE*, *39*(2, SI), 98-106. doi: 10.1559/1523040639298

Fuller, M., Seegar, W., & Schueck, L. (1998). Routes and travel rates of migrating peregrine falcons falco peregrinus and swainson's hawks buteo swainsoni in the western hemisphere. *Journal of Avian Biology*, *29*, 433-440.

GeoServer. (2013). *Wms animator.* Retrieved 2013-08-09, from http://docs.geoserver.org/stable/en/user/tutorials/animreflector.html

Gudecha, M. T. (2014). *Development of a moving object data type in a dbms* (Unpublished master's thesis). International Institute for Geo-information Science and Earth Observation (ITC).

Harrower, M., & Fabrikant, S. (2008). The role of map animation for geographic visualization. In *Geographic visualization* (pp. 49–65). John Wiley & Sons, Ltd. Retrieved from http://dx.doi.org/10.1002/9780470987643.ch4 doi: 10.1002/9780470987643.ch4

Heer, J., & Bostock, M. (2010). Declarative language design for interactive visualization. *Visualization and Computer Graphics, IEEE Transactions on*, *16*(6), 1149-1156.

i2maps. (2010). *What is i2maps?* Retrieved 2013-08-09, from http://ncg.nuim.ie/i2maps/docs/

Khronos Group. (2011). *What is webgl?* Retrieved 2013-10-08, from http://www.khronos.org/webgl/wiki/Getting_Started

Kraak, M., & Ormeling, F. (2011). *Cartography : visualization of spatial data*. Pearson Education.

Lime, S., McKenna, J., & Doyon, J.-F. (2013). *Mapfile.* Retrieved 2013-08-23, from http://www.mapserver.org/mapfile/

Lobben, A. (2003). Classification and application of cartographic animation. *The Professional Geographer*, *55*(3), 318-328.

Louden, K. (2004). *Construcción de compiladores: principios y práctica*. Thomson.

MacEachren, A. M. (1994). Time as a cartographic variable. In *Visualization in geographical information systems* (pp. 115–130). John Wiley & Sons.

Mahajan, D., Huang, F.-C., Matusik, W., Ramamoorthi, R., & Belhumeur, P. (2009, July). Moving gradients: A path-based method for plausible image interpolation. *ACM Trans. Graph.*, *28*(3), 42:1–42:11. Retrieved from http://doi.acm.org/10.1145/1531326.1531348 doi: 10.1145/1531326.1531348

Málková, M., Parus, J., Kolingerová, I., & Beneš, B. (2010). An intuitive polygon morphing. *The Visual Computer*, *26*(3), 205-215. Retrieved from http://dx.doi.org/10.1007/s00371-009-0396-3 doi: 10.1007/s00371-009-0396-3

Monmonier, M. (1990). Strategies for the visualization of geographic time-series data. *Cartographica: The International Journal for Geographic Information and Geovisualization*, *27*(1), 30-45.

Ogao, P., & Kraak, M.-J. (2002). Defining visualization operations for temporal cartographic animation design. *International Journal of Applied Earth Observation and Geoinformation*, *4*, 23-31.

Ogao, P., Ormeling, F., & Kraak, M. (2002). *Exploratory visualization of temporal geospatial data using animation* (PhD thesis).

Open Source Geospatial Foundation, O. (2013). *Gdal/ogr in python.* Retrieved 2013-10-25, from http://trac.osgeo.org/gdal/wiki/GdalOgrInPython

Peuquet, D. J. (1994). It's about time: A conceptual framework for the representation of temporal

dynamics in geographic information systems. *Annals of the Association of American Geographers*, *84*(3), 441–461. Retrieved from http://dx.doi.org/10.1111/j.1467-8306.1994.tb01869.x doi: 10.1111/j.1467-8306.1994.tb01869.x

Python Software Foundation. (2013). *Computer programming for everybody.* Retrieved 2013-10-10, from http://www.python.org/doc/essays/everybody/

Roberts, P. (2013, April 2). *Imperative vs declarative.* Retrieved 2014-02-19, from http://latentflip.com/imperative-vs-declarative/

Sayar, A. (2012). Distributed map animation services for spatiotemporal datasets. *Information Technologies and Control*, *1*, 2-8.

Schwartz, M., & Caprio, J. (2003). *North american first leaf and first bloom lilac phenology data, igbp pages/world data center for paleoclimatology data contribution series.* Retrieved 2013-12-20, from ftp://ftp.ncdc.noaa.gov/pub/data/paleo/phenology/north_america_lilac.txt

Tennent, R. (1981). *Principles of programming languages.* Englewood Cliffs etc.: Prentice-Hall International.

The pandas development team. (2013). *Pandas: Python data analysis library.* Retrieved 2013-10-25, from http://pandas.pydata.org/

UNISYS. (2014). *Hurricane/tropical data.* Retrieved 2014-01-06, from http://weather.unisys.com/hurricane/

W3C. (2012). *Synchronized multimedia.* Retrieved 2013-10-08, from http://www.w3.org/AudioVideo/

Weber, C. R. (1991). *A cartographic animation of average yearly surface temperatures for the 48 contiguous united states: 1897-1986.*

Appendix A
ASPEC-L: Objects and properties

On this appendix, we describe the objects and properties that are part of ASPEC-L. In the tables containing the description of the properties, the compulsory ones are written in normal font and the optional ones in italics.

A.1 ANIMATION OBJECT

ANIMATION object is the root element of the hierarchy of objects in the ASPEC-L files. It holds properties that apply for the animation as a whole (e.g. start time, end time, projection and output format) and contains the ***LAYER*** objects. For the description of the attributes of ***ANIMATION*** object refer to Table A.1.

A.2 LAYER OBJECT

The ***LAYER*** object contains the specification of the data source and the ***CLASS*** objects which helps to specify how to graphically represent the data in the animation. Additionally, in this object is where the type of animation should be specified. This is because for each layer a different type of animation can be applied. For the description of the attributes of ***LAYER*** object refer to Table A.2.

A.3 CLASS OBJECT

The ***CLASS*** object is aimed to define a subset of features from a ***LAYER*** object. In combination with ***LABEL*** and ***STYLE*** objects, can be used to define different symbolizations and labeling for each subsets in ***LAYER***. For the description of the attributes of the ***CLASS*** objects refer to Table A.3.

If a ***CLASS*** object is specified without the attribute ***EXPRESSION***, it is considered that any object that doesn't belong to any other ***CLASS***, belongs to it. In other words, a ***CLASS*** object without attribute ***EXPRESSION*** is considered the default class.

A.4 LABEL OBJECT

The ***LABEL*** object define how the features should be labeled, this object works over a subset defined by a ***CLASS*** object. For the description of the attributes for ***LABEL*** object refer to Table A.4.

A.5 STYLE OBJECT

The ***STYLE*** object defines a set of visual attributes to be applied to the objects within the subset defined by a ***CLASS*** object. For the description of the attributes of ***STYLE*** objects refer to Table A.5.

Table A.1 Properties for ***ANIMATION*** object.

PROPERTY	VALUE TYPE	DESCRIPTION
ADJUST_TO	String	Name of the layer to be used to auto adjust the spatial extension when ***BBOX_BEHAVIOUR*** is set to ***AUTO***.
BASE_LAYER	String	Specify a imagery provider as base layer for the animation. applicable values are [Bing | Esri | Google | OpenStreetMap].
BBOX	Extension	IF ***BBOX_BEHAVIOUR*** is set to ***FIXED***, this determines the spatial extension that will be represented in the animation. For other values of ***BBOX_BEHAVIOUR*** this property specify the initial spatial extension depicted in the animation.
BBOX_BEHAVIOUR	String	Determines how the spatial extension of the animation will behave. Applicable values are [AUTO | FIXED | USER]. ***AUTO*** means the extension will change to follow the phenomena, ***FIXED*** means that the extension is determine by the value in ***BBOX*** attribute, and ***USER*** means that the user can control de extension by performing pan and zoom.
DURATION	Duration	Specify the duration of the animation in minutes and seconds. For example 01:30, indicates an animation of one minute and thirty seconds.
END_TIME	Time	Provides temporal filtering. All the records with time stamp after ***END_TIME*** will be filter out.
FORMAT	String	Determine the output format. Applicable values are [WEBGL | SVG | FLASH].
HEIGHT	Integer	Specifies the output height in pixels.
LAYERS	Container	Object container for the collection of ***LAYER*** objects to be included in the animation.
NAME	String	Name to identify the animation, if the output template includes a placeholder for name, this value is included there.
OUTPUT_NAME	String	Specify the name for the file or folder of the produced output.
PROJECTION	Integer	EPSG code of the SRS in which the output will be produced.
START_TIME	Time	Provides temporal filtering. All the records with time stamp before ***START_TIME*** will be filter out.
THEMATIC_LEGEND	String	Indicates whether the animation should include a thematic legend or not. Applicable values are [ON | OFF].
WIDTH	Integer	Specifies the output width in pixels.

Table A.2 Properties for ***LAYER*** object.

PROPERTY	VALUE TYPE	DESCRIPTION
ANIMATION_TYPE	String	Specify the type of animation to be created. For vector data, the applicable values are [EXISTENTIAL | STEPWISE | INTERPOLATED], and for raster data [DIRECT | FADE | INTERPOLATED].
BANDS	String	List of 3 integer numbers separated by commas that indicate the bands of a raster source to be processed. .
CLASSES	Container	Object container for the collection of ***CLASS*** objects.
CONNECTION_STRING	String	For data sources of type spatial database, this value indicates the parameters to connect to the server.
DATA	String	Specify the dataset to be used. In the case of spatial databases, it specify name of the table to be used; for OWSes it specify the layer on the service; and for file datasources, it indicates the location of the file.
DATASOURCE	String	Specify the type of datasource. The applicable values are [WFS | WMS | POSTGIS | MYSQL | SQLSERVER | ORACLE | FILE].
DURATION	Decimal	In existential animation, this value specify the lifetime of the features in seconds of animation time.
FIELD_GEOMETRY	String	Specify the field that contains the geometry in the dataset.
FIELD_GROUP_BY	String	This value is used when the animation to be created requires to group the data, e.g. ***INTERPOLATED*** animation.
FIELD_TIME	String	Specify the field that contains the timestamps in the dataset. For existential animation, this field determines when the feature start its lifetime, can be set to ***START***, to indicate that the feature will be shown from the start of the animation.
FIELD_TIME_2	String	In existential animation, this value specify the end of life time of features. Can be replaced by ***END***, to indicate that the feature will last till the end of the animation.
FILTER	String	Provides attribute filtering. It is a SQL where clause.
NAME	String	Specify the layer's name. If the output template includes a placeholder for layer names, this value is shown there. Additionally it is use in ***ADJUST_TO*** attribute in ***ANIMATION*** object.
PROJECTION	Integer	EPSG code of the SRS of the dataset.
STATUS	String	Indicates whether the layer should be or not include in the animation. Applicable values [ON | OFF].
TYPE	String	Specify the data type in the layer. Applicable values [POINT | LINE | POLYGON | RASTER].
URL	String	Specify the URI of an OWS to be used as datasource.

Table A.3 Properties for ***CLASS*** object.

PROPERTY	VALUE TYPE	DESCRIPTION
EXPRESSION	String	Logical expression to select the subset of features that belong to the class.
LABEL	Container	Container for label object, used to specify how to label the features of the class.
NAME	String	Name that identify the class, if the output template includes a placeholder for the name of the classes, this value is shown there.
STATUS	String	Indicates whether the class should or not be taken in account to produce the animation. Applicable values [ON | OFF].
STYLES	Container	Container for ***STYLE*** objects.

Table A.4 Properties for ***LABEL*** object.

PROPERTY	VALUE TYPE	DESCRIPTION
COLOR	Color	Specify the color in RGBA format for the labels.
FIELD_LABEL	String	Specify the field to be used for labeling.
FONT	String	Name of the font to be used.
OUTLINE_COLOR	Color	Specify the color in RGBA format for the labels outline.
SIZE	String	Arithmetical expression to specify the size of the font in pixels.

Table A.5 Properties for ***STYLE*** object.

PROPERTY	VALUE TYPE	DESCRIPTION
COLOR	Color	Specify the color in RGBA format to be applied in the visual representation of the features as fill color.
OUTLINE_COLOR	Color	Specify the color in RGBA format to be applied in the visual representation of the features as outline color.
OUTLINE_WIDTH	String	Arithmetical expression to specify the width of lines in pixels.
SIZE	String	Arithmetical expression to specify the size in pixels for symbols.
SYMBOL	String	Specify the name of an image or predefined symbol to be used to represent point features or as pattern for lines and polygons.

Appendix B
ASPEC-L specification code for application cases

B.1 ASPEC-L SPECIFICATIONS FOR LILAC BLOOMING

B.1.1 Lilac blooming: first leaf and bloom 1968

Listing B.1: CZML code for Lilac blooming - first leaf and bloom 1968.

```
ANIMATION
    NAME "Lilac blooming 1968"
    START_TIME "19680101T00:00:00"
    END_TIME "19680630T00:00:00"
    DURATION 02:00
    FORMAT "WEBGL"
    BBOX -124.36 29.53 -52.78 49.25
    BBOX_BEHAVIOUR "FIXED"
    OUTPUT_NAME "lilac01"

    LAYER
        DATASOURCE "WFS"
        TYPE "POINT"
        DATA "lilac"
        URL "http://localhost:8081/cgi-bin/mapserv.exe?map=c:/ms4w/apps/awm/awm.map&"
        ANIMATION_TYPE "EXISTENTIAL"
        FIELD_TIME "bloom"
        FIELD_TIME_2 "END"
        CLASS
            COLOR 255 0 0 100
            SIZE 10.0
            OUTLINE_COLOR 0 0 0 100
            OUTLINE_WIDTH 2
        END
    END

    LAYER
        DATASOURCE "WFS"
        TYPE "POINT"
        DATA "lilac"
        URL "http://localhost:8081/cgi-bin/mapserv.exe?map=c:/ms4w/apps/awm/awm.map&"
        ANIMATION_TYPE "EXISTENTIAL"
        FIELD_TIME "leaf"
        FIELD_TIME_2 "END"
        CLASS
            COLOR 255 255 0 100
            SIZE 8.0
            OUTLINE_COLOR 0 0 0 100
```

```
            OUTLINE_WIDTH 2
        END
    END
END
```

B.1.2 Lilac blooming: first leaf and bloom 1970 - 2003

Listing B.2: CZML code for Lilac blooming - first leaf and bloom 1970 - 2003.

```
ANIMATION
    NAME "Lilac blooming 1970-2003"
    START_TIME "19700101T00:00:00"
    DURATION 02:00
    FORMAT "WEBGL"
    BBOX -124.36 29.53 -52.78 49.25
    BBOX_BEHAVIOUR "FIXED"
    OUTPUT_NAME "lilac02"

    LAYER
        DATASOURCE "WFS"
        TYPE "POINT"
        DATA "lilac"
        URL "http://localhost:8081/cgi-bin/mapserv.exe?map=c:/ms4w/apps/awm/awm.map&"
        ANIMATION_TYPE "EXISTENTIAL"
        FIELD_TIME "bloom"
        DURATION 2
        CLASS
            COLOR 255 0 0 100
            SIZE 8.0
            OUTLINE_COLOR 0 0 0 100
            OUTLINE_WIDTH 2
        END
    END

    LAYER
        DATASOURCE "WFS"
        TYPE "POINT"
        DATA "lilac"
        URL "http://localhost:8081/cgi-bin/mapserv.exe?map=c:/ms4w/apps/awm/awm.map&"
        ANIMATION_TYPE "EXISTENTIAL"
        FIELD_TIME "leaf"
        DURATION 2
        CLASS
            COLOR 255 255 0 255
            SIZE 4.0
            OUTLINE_COLOR 0 0 0 255
            OUTLINE_WIDTH 2
        END
    END
END
```

B.1.3 Lilac blooming: first leaf and bloom whole dataset

Listing B.3: CZML code for Lilac blooming - first leaf and bloom whole dataset.

```
ANIMATION
    NAME "Lilac blooming"
    DURATION 02:00
    FORMAT "WEBGL"
    BBOX -124.36 29.53 -52.78 49.25
    BBOX_BEHAVIOUR "FIXED"
    OUTPUT_NAME "lilac03"

    LAYER
        DATASOURCE "WFS"
        TYPE "POINT"
        DATA "lilac"
        URL "http://localhost:8081/cgi-bin/mapserv.exe?map=c:/ms4w/apps/awm/awm.map&"
        ANIMATION_TYPE "EXISTENTIAL"
        FIELD_TIME "bloom"
        DURATION 1
        CLASS
            COLOR 255 0 0 100
            SIZE 8.0
            OUTLINE_COLOR 0 0 0 100
            OUTLINE_WIDTH 2
        END
    END

    LAYER
        DATASOURCE "WFS"
        TYPE "POINT"
        DATA "lilac"
        URL "http://localhost:8081/cgi-bin/mapserv.exe?map=c:/ms4w/apps/awm/awm.map&"
        ANIMATION_TYPE "EXISTENTIAL"
        FIELD_TIME "leaf"
        DURATION 1
        CLASS
            COLOR 255 255 0 100
            SIZE 4.0
            OUTLINE_COLOR 0 0 0 100
            OUTLINE_WIDTH 2
        END
    END
END
```

The other three animations for Lilac blooming application case uses the same specification from Listing B.3. The only modification is the value of ***DATA*** field in ***LAYER*** object to consume 3 simulated datasets.

B.2 ASPEC-L SPECIFICATIONS FOR HAWKS MIGRATION

B.2.1 Swainson's Hawk migration 1996-1997: one symbolization class and fixed spatial extent

Listing B.4: CZML code for Swainson's Hawk migration 1996-1997 - one symbolization class and fixed spatial extent.

```
ANIMATION
    NAME "Hawks movement"
    START_TIME "1996-07-24T00:00:00"
    END_TIME "1997-06-29T02:49:00"
    DURATION 01:00
    FORMAT "WEBGL"
    BBOX -122 -39 -56 54
    BBOX_BEHAVIOUR "FIXED"
    OUTPUT_NAME "hawks01"

    LAYER
        NAME "hawks"
        DATASOURCE "WFS"
        TYPE "POINT"
        DATA "hawks"
        URL "http://localhost:8081/cgi-bin/mapserv.exe?map=c:/ms4w/apps/awm/awm.map&"
        ANIMATION_TYPE "INTERPOLATED"
        FIELD_GROUP_BY "ind_ident"
        FIELD_TIME "timestamp"
        CLASS
            COLOR 0 0 255 100
            SIZE 6.0
            OUTLINE_COLOR 0 0 0 100
            OUTLINE_WIDTH 2
        END
    END
END
```

B.2.2 Swainson's Hawk migration 1996-1997: two symbolization classes and fixed spatial extent

Listing B.5: CZML code for Swainson's Hawk migration 1996-1997 - two symbolization classes and fixed spatial extent.

```
ANIMATION
    NAME "Hawks movement"
    START_TIME "1996-07-24T00:00:00"
    END_TIME "1997-06-29T02:49:00"
    DURATION 01:00
    FORMAT "WEBGL"
    BBOX -122 -39 -56 54
    BBOX_BEHAVIOUR "FIXED"
    OUTPUT_NAME "hawks02"

    LAYER
        NAME "hawks"
        DATASOURCE "WFS"
        TYPE "POINT"
        DATA "hawks"
        URL "http://localhost:8081/cgi-bin/mapserv.exe?map=c:/ms4w/apps/awm/awm.map&"
        ANIMATION_TYPE "INTERPOLATED"
        FIELD_GROUP_BY "ind_ident"
        FIELD_TIME "timestamp"
        CLASS
```

```
21              EXPRESSION "'ind_ident' in ('SW1', 'SW3', 'SW4', 'SW5', 'SW6', 'SW7', 'SW8', '
                    SW9', 'SW10', 'SW11', 'SW12', 'SW15', 'SW16', 'SW17', 'SW18', 'SW19', '
                    SW20', 'SW21', 'SW22', 'SW23')"
22              COLOR 0 0 255 100
23              SIZE 6.0
24              OUTLINE_COLOR 0 0 0 100
25              OUTLINE_WIDTH 2
26          END
27          CLASS
28              EXPRESSION "'ind_ident' in ('SW24', 'SW25', 'SW26', 'SW27', 'SW28', 'SW29', '
                    SW30', 'SW31', 'SW32', 'SW33', 'SW34', 'SW35', 'SW36', 'SW37', 'SW38', '
                    SW39', 'SW40', 'SW41', 'SW42', 'SW43', 'SW44', 'SW45', 'SW46', 'SW47', '
                    SW48')"
29              COLOR 255 0 0 100
30              SIZE 6.0
31              OUTLINE_COLOR 0 0 0 100
32              OUTLINE_WIDTH 2
33          END
34      END
35  END
```

B.2.3 Swainson's Hawk migration 1996-1997: four symbolization classes and fixed spatial extent

Listing B.6: CZML code for Swainson's Hawk migration 1996-1997 - four symbolization classes and fixed spatial extent.

```
1   ANIMATION
2       NAME "Hawks movement"
3       START_TIME "1996-07-24T00:00:00"
4       END_TIME "1997-06-29T02:49:00"
5       DURATION 01:00
6       FORMAT "WEBGL"
7       BBOX -122 -39 -56 54
8       BBOX_BEHAVIOUR "FIXED"
9       OUTPUT_NAME "hawks03"
10
11      LAYER
12          NAME "hawks"
13          DATASOURCE "WFS"
14          TYPE "POINT"
15          DATA "hawks"
16          URL "http://localhost:8081/cgi-bin/mapserv.exe?map=c:/ms4w/apps/awm/awm.map&"
17          ANIMATION_TYPE "INTERPOLATED"
18          FIELD_GROUP_BY "ind_ident"
19          FIELD_TIME "timestamp"
20          CLASS
21              EXPRESSION "'ind_ident' in ('SW1', 'SW3', 'SW4', 'SW5', 'SW6', 'SW7', 'SW8', '
                    SW9', 'SW10', 'SW11')"
22              COLOR 0 0 255 100
23              SIZE 6.0
24              OUTLINE_COLOR 0 0 0 100
25              OUTLINE_WIDTH 2
26          END
27          CLASS
```

```
28              EXPRESSION "'ind_ident' in ('SW12', 'SW15', 'SW16', 'SW17', 'SW18', 'SW19', '
                    SW20', 'SW21', 'SW22', 'SW23')"
29              COLOR 0 255 0 100
30              SIZE 6.0
31              OUTLINE_COLOR 0 0 0 100
32              OUTLINE_WIDTH 2
33          END
34          CLASS
35              EXPRESSION "'ind_ident' in ('SW24', 'SW25', 'SW26', 'SW27', 'SW28', 'SW29', '
                    SW30', 'SW31', 'SW32', 'SW33', 'SW34')"
36              COLOR 255 0 0 100
37              SIZE 6.0
38              OUTLINE_COLOR 0 0 0 100
39              OUTLINE_WIDTH 2
40          END
41          CLASS
42              EXPRESSION "'ind_ident' in ('SW35', 'SW36', 'SW37','SW38', 'SW39', 'SW40', '
                    SW41', 'SW42', 'SW43', 'SW44', 'SW45', 'SW46', 'SW47', 'SW48')"
43              COLOR 0 255 255 100
44              SIZE 6.0
45              OUTLINE_COLOR 0 0 0 100
46              OUTLINE_WIDTH 2
47          END
48      END
49  END
```

B.2.4 Swainson's Hawk migration 1996-1997: eight symbolization classes and fixed spatial extent

Listing B.7: CZML code for Swainson's Hawk migration 1996-1997 - eight symbolization classes and fixed spatial extent.

```
1   ANIMATION
2       NAME "Hawks movement"
3       START_TIME "1996-07-24T00:00:00"
4       END_TIME "1997-06-29T02:49:00"
5       DURATION 01:00
6       FORMAT "WEBGL"
7       BBOX -122 -39 -56 54
8       BBOX_BEHAVIOUR "FIXED"
9       OUTPUT_NAME "hawks04"
10
11      LAYER
12          NAME "hawks"
13          DATASOURCE "WFS"
14          TYPE "POINT"
15          DATA "hawks"
16          URL "http://localhost:8081/cgi-bin/mapserv.exe?map=c:/ms4w/apps/awm/awm.map&"
17          ANIMATION_TYPE "INTERPOLATED"
18          FIELD_GROUP_BY "ind_ident"
19          FIELD_TIME "timestamp"
20          CLASS
21              EXPRESSION "'ind_ident' in ('SW1', 'SW3', 'SW4', 'SW5', 'SW6')"
22              COLOR 0 0 255 100
23              SIZE 6.0
```

```
            OUTLINE_COLOR 0 0 0 100
            OUTLINE_WIDTH 2
        END
        CLASS
            EXPRESSION "'ind_ident' in ('SW7', 'SW8', 'SW9', 'SW10', 'SW11')"
            COLOR 0 100 255 100
            SIZE 6.0
            OUTLINE_COLOR 0 0 0 100
            OUTLINE_WIDTH 2
        END
        CLASS
            EXPRESSION "'ind_ident' in ('SW12', 'SW15', 'SW16', 'SW17', 'SW18')"
            COLOR 0 255 0 100
            SIZE 6.0
            OUTLINE_COLOR 0 0 0 100
            OUTLINE_WIDTH 2
        END
        CLASS
            EXPRESSION "'ind_ident' in ('SW19', 'SW20', 'SW21', 'SW22', 'SW23')"
            COLOR 0 255 100 100
            SIZE 6.0
            OUTLINE_COLOR 0 0 0 100
            OUTLINE_WIDTH 2
        END
        CLASS
            EXPRESSION "'ind_ident' in ('SW24', 'SW25', 'SW26', 'SW27', 'SW28', 'SW29')"
            COLOR 255 0 0 100
            SIZE 6.0
            OUTLINE_COLOR 0 0 0 100
            OUTLINE_WIDTH 2
        END
        CLASS
            EXPRESSION "'ind_ident' in ('SW30', 'SW31', 'SW32', 'SW33', 'SW34')"
            COLOR 255 0 100 100
            SIZE 6.0
            OUTLINE_COLOR 0 0 0 100
            OUTLINE_WIDTH 2
        END
        CLASS
            EXPRESSION "'ind_ident' in ('SW35', 'SW36', 'SW37', 'SW38', 'SW39', 'SW40')"
            COLOR 0 255 255 100
            SIZE 6.0
            OUTLINE_COLOR 0 0 0 100
            OUTLINE_WIDTH 2
        END
        CLASS
            EXPRESSION "'ind_ident' in ('SW41', 'SW42', 'SW43', 'SW44', 'SW45', 'SW46', '
                SW47', 'SW48')"
            COLOR 100 255 255 100
            SIZE 6.0
            OUTLINE_COLOR 0 0 0 100
            OUTLINE_WIDTH 2
        END
    END
END
```

B.2.5 Swainson's Hawk migration 1996-1997: sixteen symbolization classes and fixed spatial extent

Listing B.8: CZML code for Swainson's Hawk migration 1996-1997 - sixteen symbolization classes and fixed spatial extent.

```
ANIMATION
    NAME "Hawks movement"
    START_TIME "1996-07-24T00:00:00"
    END_TIME "1997-06-29T02:49:00"
    DURATION 01:00
    FORMAT "WEBGL"
    BBOX -122 -39 -56 54
    BBOX_BEHAVIOUR "FIXED"
    OUTPUT_NAME "hawks05"

    LAYER
        NAME "hawks"
        DATASOURCE "WFS"
        TYPE "POINT"
        DATA "hawks"
        URL "http://localhost:8081/cgi-bin/mapserv.exe?map=c:/ms4w/apps/awm/awm.map&"
        ANIMATION_TYPE "INTERPOLATED"
        FIELD_GROUP_BY "ind_ident"
        FIELD_TIME "timestamp"
        CLASS
            EXPRESSION "'ind_ident' in ('SW1', 'SW3', 'SW4')"
            COLOR 0 0 255 100
            SIZE 6.0
            OUTLINE_COLOR 0 0 0 100
            OUTLINE_WIDTH 2
        END
        CLASS
            EXPRESSION "'ind_ident' in ('SW5', 'SW6')"
            COLOR 0 200 255 100
            SIZE 6.0
            OUTLINE_COLOR 0 0 0 100
            OUTLINE_WIDTH 2
        END
        CLASS
            EXPRESSION "'ind_ident' in ('SW7', 'SW8', 'SW9')"
            COLOR 0 100 255 100
            SIZE 6.0
            OUTLINE_COLOR 0 0 0 100
            OUTLINE_WIDTH 2
        END
        CLASS
            EXPRESSION "'ind_ident' in ('SW10', 'SW11')"
            COLOR 200 100 255 100
            SIZE 6.0
            OUTLINE_COLOR 0 0 0 100
            OUTLINE_WIDTH 2
        END
        CLASS
            EXPRESSION "'ind_ident' in ('SW12', 'SW15', 'SW16')"
            COLOR 0 255 0 100
```

```
            SIZE 6.0
            OUTLINE_COLOR 0 0 0 100
            OUTLINE_WIDTH 2
        END
        CLASS
            EXPRESSION "'ind_ident' in ('SW17', 'SW18')"
            COLOR 0 255 200 100
            SIZE 6.0
            OUTLINE_COLOR 0 0 0 100
            OUTLINE_WIDTH 2
        END
        CLASS
            EXPRESSION "'ind_ident' in ('SW19', 'SW20', 'SW21')"
            COLOR 0 255 100 100
            SIZE 6.0
            OUTLINE_COLOR 0 0 0 100
            OUTLINE_WIDTH 2
        END
        CLASS
            EXPRESSION "'ind_ident' in ('SW22', 'SW23')"
            COLOR 200 255 100 100
            SIZE 6.0
            OUTLINE_COLOR 0 0 0 100
            OUTLINE_WIDTH 2
        END
        CLASS
            EXPRESSION "'ind_ident' in ('SW24', 'SW25', 'SW26')"
            COLOR 255 0 0 100
            SIZE 6.0
            OUTLINE_COLOR 0 0 0 100
            OUTLINE_WIDTH 2
        END
        CLASS
            EXPRESSION "'ind_ident' in ('SW27', 'SW28', 'SW29')"
            COLOR 255 0 200 100
            SIZE 6.0
            OUTLINE_COLOR 0 0 0 100
            OUTLINE_WIDTH 2
        END
        CLASS
            EXPRESSION "'ind_ident' in ('SW30', 'SW31', 'SW32')"
            COLOR 255 0 100 100
            SIZE 6.0
            OUTLINE_COLOR 0 0 0 100
            OUTLINE_WIDTH 2
        END
        CLASS
            EXPRESSION "'ind_ident' in ('SW33', 'SW34')"
            COLOR 255 200 100 100
            SIZE 6.0
            OUTLINE_COLOR 0 0 0 100
            OUTLINE_WIDTH 2
        END
        CLASS
            EXPRESSION "'ind_ident' in ('SW35', 'SW36', 'SW37')"
```

```
            COLOR 0 255 255 100
            SIZE 6.0
            OUTLINE_COLOR 0 0 0 100
            OUTLINE_WIDTH 2
        END
        CLASS
            EXPRESSION "'ind_ident' in ('SW38', 'SW39', 'SW40')"
            COLOR 200 255 255 100
            SIZE 6.0
            OUTLINE_COLOR 0 0 0 100
            OUTLINE_WIDTH 2
        END
        CLASS
            EXPRESSION "'ind_ident' in ('SW41', 'SW42', 'SW43', 'SW44')"
            COLOR 100 255 255 100
            SIZE 6.0
            OUTLINE_COLOR 0 0 0 100
            OUTLINE_WIDTH 2
        END
        CLASS
            EXPRESSION "'ind_ident' in ('SW45', 'SW46', 'SW47', 'SW48')"
            COLOR 100 200 0 100
            SIZE 6.0
            OUTLINE_COLOR 0 0 0 100
            OUTLINE_WIDTH 2
        END
    END
END
```

B.2.6 Swainson's Hawk migration 1996-1997: one symbolization class and auto-adjustment spatial extent

Listing B.9: CZML code for Swainson's Hawk migration 1996-1997 - one symbolization class and auto-adjustment spatial extent.

```
ANIMATION
    NAME "Hawks movement"
    START_TIME "1996-07-24T00:00:00"
    END_TIME "1997-06-29T02:49:00"
    DURATION 01:00
    FORMAT "WEBGL"
    BBOX -122 -39 -56 54
    BBOX_BEHAVIOUR "AUTO"
    OUTPUT_NAME "hawks05"
    ADJUST_TO "hawks"
    WIDTH 1024
    HEIGHT 768

    LAYER
        NAME "hawks"
        DATASOURCE "WFS"
        TYPE "POINT"
        DATA "hawks"
        URL "http://localhost:8081/cgi-bin/mapserv.exe?map=c:/ms4w/apps/awm/awm.map&"
        ANIMATION_TYPE "INTERPOLATED"
```

```
        FIELD_GROUP_BY "ind_ident"
        FIELD_TIME "timestamp"
        CLASS
            COLOR 0 0 255 100
            SIZE 6.0
            OUTLINE_COLOR 0 0 0 100
            OUTLINE_WIDTH 2
        END
    END
END
```

B.3 ASPEC-L SPECIFICATIONS FOR HURRICANE MOVEMENT

B.3.1 Hurricane Ivan 2004: Stepwise animation

Listing B.10: CZML code for Hurricane Ivan 2004 - Stepwise animation.

```
ANIMATION
    NAME "Hurrican Ivan 2004"
    DURATION "00:30"
    FORMAT "WEBGL"
    BBOX -95.0 7.0 -26.5 39.0
    BBOX_BEHAVIOUR "USER"
    OUTPUT_NAME "hurricane01"

    LAYER
        DATASOURCE "WFS"
        TYPE "POINT"
        DATA "hurricanes"
        URL "http://localhost:8081/cgi-bin/mapserv.exe?
            map=c:/ms4w/apps/awm/awm.map&"
        ANIMATION_TYPE "STEPWISE"
        FIELD_TIME "date_time"
        FIELD_GROUP_BY "name"
        FILTER "year_no = '2004-9'"

        CLASS
            COLOR 255 0 0 100
            SIZE 8
            OUTLINE_COLOR 0 0 0 100
            OUTLINE_WIDTH 2
        END
    END
END
```

B.3.2 Hurricane Ivan 2004: animation with interpolation for location

Listing B.11: CZML code for Hurricane Ivan 2004 - animation with interpolation for location.

```
ANIMATION
    NAME "Hurrican Ivan 2004"
    DURATION "00:30"
```

```
    FORMAT "WEBGL"
    BBOX -95.0 7.0 -26.5 39.0
    BBOX_BEHAVIOUR "USER"
    OUTPUT_NAME "hurricane02"

    LAYER
        DATASOURCE "WFS"
        TYPE "POINT"
        DATA "hurricanes"
        URL "http://localhost:8081/cgi-bin/mapserv.exe?
             map=c:/ms4w/apps/awm/awm.map&"
        ANIMATION_TYPE "INTERPOLATED"
        FIELD_TIME "date_time"
        FIELD_GROUP_BY "name"
        FILTER "year_no = '2004-9'"

        CLASS
            COLOR 255 0 0 100
            SIZE 8
            OUTLINE_COLOR 0 0 0 100
            OUTLINE_WIDTH 2
        END
    END
END
```

B.3.3 Hurricane Ivan 2004: animation with interpolation for location and size

Listing B.12: CZML code for Hurricane Ivan 2004 - animation with interpolation for location and size.

```
ANIMATION
    NAME "Hurrican Ivan 2004"
    DURATION "00:30"
    FORMAT "WEBGL"
    BBOX -95.0 7.0 -26.5 39.0
    BBOX_BEHAVIOUR "USER"
    OUTPUT_NAME "hurricane03"

    LAYER
        DATASOURCE "WFS"
        TYPE "POINT"
        DATA "hurricanes"
        URL "http://localhost:8081/cgi-bin/mapserv.exe?map=c:/ms4w/apps/awm/awm.map&"
        ANIMATION_TYPE "INTERPOLATED"
        FIELD_TIME "date_time"
        FIELD_GROUP_BY "name"
        FILTER "year_no = '2004-9'"

        CLASS
            COLOR 255 0 0 100
            SIZE "wspeed / 2"
            OUTLINE_COLOR 0 0 0 100
            OUTLINE_WIDTH 2
        END
```

```
    END
END
```

B.3.4 Hurricane Ivan 2004: animation with interpolation for location and color

Listing B.13: CZML code for Hurricane Ivan 2004 - animation with interpolation for location and color.

```
ANIMATION
    NAME "Hurrican Ivan 2004"
    DURATION "00:30"
    FORMAT "WEBGL"
    BBOX -95.0 7.0 -26.5 39.0
    BBOX_BEHAVIOUR "USER"
    OUTPUT_NAME "hurricane04"

    LAYER
        DATASOURCE "WFS"
        TYPE "POINT"
        DATA "hurricanes"
        URL "http://localhost:8081/cgi-bin/mapserv.exe?map=c:/ms4w/apps/awm/awm.map&"
        ANIMATION_TYPE "INTERPOLATED"
        FIELD_TIME "date_time"
        FIELD_GROUP_BY "name"
        FILTER "year_no = '2004-9'"

        CLASS
            COLOR 0 255 0 100
            SIZE 8
            OUTLINE_COLOR 0 0 0 100
            OUTLINE_WIDTH 2
        END

        CLASS
            EXPRESSION "wspeed > 70"
            COLOR 255 255 0 100
            SIZE 8
            OUTLINE_COLOR 0 0 0 100
            OUTLINE_WIDTH 2
        END

        CLASS
            EXPRESSION "wspeed > 100"
            COLOR 255 0 0 100
            SIZE 8
            OUTLINE_COLOR 0 0 0 100
            OUTLINE_WIDTH 2
        END
    END
END
```

B.3.5 Hurricane Ivan 2004: animation with interpolation for location, size and color

Listing B.14: CZML code for Hurricane Ivan 2004 - animation with interpolation for location, size and color.

```
ANIMATION
    NAME "Hurrican Ivan 2004"
    DURATION "00:30"
    FORMAT "WEBGL"
    BBOX -95.0 7.0 -26.5 39.0
    BBOX_BEHAVIOUR "USER"
    OUTPUT_NAME "hurricane05"

    LAYER
        DATASOURCE "WFS"
        TYPE "POINT"
        DATA "hurricanes"
        URL "http://localhost:8081/cgi-bin/mapserv.exe?
             map=c:/ms4w/apps/awm/awm.map&"
        ANIMATION_TYPE "INTERPOLATED"
        FIELD_TIME "date_time"
        FIELD_GROUP_BY "name"
        FILTER "year_no = '2004-9'"

        CLASS
            COLOR 0 255 0 100
            SIZE "wspeed / 2"
            OUTLINE_COLOR 0 0 0 100
            OUTLINE_WIDTH 2
        END

        CLASS
            EXPRESSION "wspeed > 70"
            COLOR 255 255 0 100
            SIZE "wspeed / 2"
            OUTLINE_COLOR 0 0 0 100
            OUTLINE_WIDTH 2
        END

        CLASS
            EXPRESSION "wspeed > 100"
            COLOR 255 0 0 100
            SIZE "wspeed / 2"
            OUTLINE_COLOR 0 0 0 100
            OUTLINE_WIDTH 2
        END
    END
END
```

Appendix C
CZML code for vector animation

C.1 EXISTENTIAL ANIMATION WITH POINT VECTOR DATA

Listing C.1: CZML code for existential animation with point data.

```
[
    {
      "id":"document",
      "clock":{
        "interval":"19680101T00:00:00/19680630T00:00:00",
        "currentTime":"19680101T00:00:00",
        "multiplier":130320.0,
        "range":"LOOP_STOP",
        "step":"SYSTEM_CLOCK_MULTIPLIER"
      }
    },
   {
    "id": "0",
    "availability": "1968-02-13T00:00:00/1968-06-30T00:00:00",
    "point": {
      "color": {
        "rgba": [
          255,0,0,255
        ]
      },
      "outlineColor": {
        "rgba": [
          0,0,0,255
        ]
      },
      "outlineWidth": {
        "number": 2
      },
      "pixelSize":{
        "number": 10.0
      }
    },
    "position": {
      "cartographicDegrees": [
        -117.15,33.21,0
      ]
    }
  },

  ...
```

```
41
42    {
43      "id": "1188",
44      "availability": "1968-06-12T00:00:00/1968-06-30T00:00:00",
45      "point": {
46        "color": {
47          "rgba": [
48            255,255,0,255
49          ]
50        },
51        "outlineColor": {
52          "rgba": [
53            0,0,0,255
54          ]
55        },
56        "outlineWidth": {
57          "number": 2
58        },
59        "pixelSize":{
60          "number": 8.0
61        }
62      },
63      "position": {
64        "cartographicDegrees": [
65          -111.06,44.39,0
66        ]
67      }
68    }
69  ]
```

C.2 STEPWISE ANIMATION WITH POINT VECTOR DATA

Listing C.2: CZML code for stepwise animation with point data.

```
1   [
2       {
3         "id":"document",
4         "clock":{
5           "interval":"2004-09-02T18:00:00/2004-09-24T06:00:00",
6           "currentTime":"2004-09-02T18:00:00",
7           "multiplier":61920.0,
8           "range":"LOOP_STOP",
9           "step":"SYSTEM_CLOCK_MULTIPLIER"
10        }
11      },
12     {
13      "id": "IVAN-1",
14      "availability": "2004-09-02T18:00:00/2004-09-03T00:00:00",
15      "point": {
16        "color": {
17          "rgba": [
18            255,0,0,255
19          ]
```

```
      },
      "outlineColor": {
        "rgba": [
          0,0,0,255
        ]
      },
      "outlineWidth": {
        "number": 2
      },
      "pixelSize":{
        "number": 8
      }
    },
    "position": {
      "cartographicDegrees": [
        -27.6,9.7,0
      ]
    }
  },

  ...

   {
    "id": "IVAN-93",
    "availability": "2004-09-24T02:00:00/2004-09-24T06:00:00",
    "point": {
      "color": {
        "rgba": [
          255,0,0,255
        ]
      },
      "outlineColor": {
        "rgba": [
          0,0,0,255
        ]
      },
      "outlineWidth": {
        "number": 2
      },
      "pixelSize":{
        "number": 8
      }
    },
    "position": {
      "cartographicDegrees": [
        -93.6,29.8,0
      ]
    }
  }
]
```

C.3 INTERPOLATED ANIMATION WITH POINT VECTOR DATA

Listing C.3: CZML code for interpolated animation with point data.

```
[
    {
      "id":"document",
      "clock":{
        "interval":"2004-09-02T18:00:00/2004-09-24T06:00:00",
        "currentTime":"2004-09-02T18:00:00",
        "multiplier":61920.0,
        "range":"LOOP_STOP",
        "step":"SYSTEM_CLOCK_MULTIPLIER"
      }
    },
   {
    "id": "IVAN",
    "availability": "2004-09-02T18:00:00/2004-09-24T06:00:00",
    "point": {
      "color": {
        "interpolationAlgorithm":"LAGRANGE",
        "interpolationDegree":1,
        "epoch":"2004-09-02T18:00:00",
        "rgba": [
          "2004-09-02T18:00:00",0,255,0,255,
          "2004-09-05T06:00:00",0,255,0,255,
          "2004-09-05T12:00:00",255,255,0,255,

          ...

          "2004-09-16T06:00:00",255,0,0,255,
          "2004-09-16T12:00:00",0,255,0,255,
          "2004-09-24T06:00:00",0,255,0,255
        ]
      },
      "outlineColor": {
        "rgba": [
          0,0,0,255
        ]
      },
      "outlineWidth": {
        "number": 2
      },
      "pixelSize":{
        "interpolationAlgorithm":"LAGRANGE",
        "interpolationDegree":1,
        "epoch":"2004-09-02T18:00:00",
        "number": [
          "2004-09-02T18:00:00",12.5,
          "2004-09-03T00:00:00",15.0,
          "2004-09-03T00:00:00",15.0,

          ...

          "2004-09-24T02:00:00",15.0,
          "2004-09-24T06:00:00",12.5,
          "2004-09-24T06:00:00",12.5
```

```
        ]
      }
    },
    "position": {
      "interpolationAlgorithm": "LAGRANGE",
      "interpolationDegree":1,
      "epoch": "2004-09-02T18:00:00",
      "cartographicDegrees": [
        "2004-09-02T18:00:00",-27.6,9.7,0,
        "2004-09-03T00:00:00",-28.7,9.7,0,
        "2004-09-03T06:00:00",-30.3,9.7,0,

        ...

        "2004-09-24T00:00:00",-93.2,29.6,0,
        "2004-09-24T02:00:00",-93.6,29.8,0,
        "2004-09-24T06:00:00",-94.2,30.1,0
      ]
    }
  }
]
```

C.4 EXISTENTIAL ANIMATION WITH LINE VECTOR DATA

Listing C.4: CZML code for existential animation with line data.

```
[
    {
      "id":"document",
      "clock":{
        "interval":"2009-05-01T12:00:00/2009-05-01T14:00:00",
        "currentTime":"2009-05-01T12:00:00",
        "multiplier":720.0,
        "range":"LOOP_STOP",
        "step":"SYSTEM_CLOCK_MULTIPLIER"
      }
    },
   {
    "id": "0",
    "availability": "2009-05-01T12:00:00/2009-05-01T14:00:00",
    "polyline": {
      "color": {
        "rgba": [
          0,0,0,255.0
        ]
      },
      "width": 3,
      "show":true
    },
    "vertexPositions":{
      "cartographicDegrees":[
          6.902388,52.27064,0,6.905297,52.278141,0,6.90943,52.280743,0,6.915859,
          52.285871,0,6.918767,52.286177,0,6.918767,52.286177,0
```

```
      ]
    }
  },

  ...

  {
    "id": "4",
    "availability": "2009-05-01T14:00:00/2009-05-01T14:00:00",
    "polyline": {
      "color": {
        "rgba": [
          0,0,0,255.0
        ]
      },
      "width": 3,
      "show":true
    },
    "vertexPositions":{
      "cartographicDegrees":[
          6.924508,52.241403,0,6.925196,52.264823,0,6.933922,52.271099,0,6.947851,
          52.277299,0,6.961398,52.279519,0,6.961398,52.279519,0
      ]
    }
  }
]
```

C.5 STEPWISE ANIMATION WITH LINE VECTOR DATA

Listing C.5: CZML code for stepwise animation with line data.

```
[
    {
      "id":"document",
      "clock":{
        "interval":"2009-05-01T12:00:00/2009-05-01T14:00:00",
        "currentTime":"2009-05-01T12:00:00",
        "multiplier":720.0,
        "range":"LOOP_STOP",
        "step":"SYSTEM_CLOCK_MULTIPLIER"
      }
    },
   {
    "id": "1-2",
    "availability": "2009-05-01T12:00:00/2009-05-01T12:30:00",
    "polyline": {
      "color": {
        "rgba": [
          0,0,0,255.0
        ]
      },
      "width": 3,
      "show":true
```

```
    },
    "vertexPositions":{
      "cartographicDegrees":[
          6.902388,52.27064,0,6.905297,52.278141,0,6.90943,52.280743,0,6.915859,
          52.285871,0,6.918767,52.286177,0,6.918767,52.286177,0
      ]
    }
  },

  ...

   {
    "id": "1-5",
    "availability": "2009-05-01T13:30:00/2009-05-01T14:00:00",
    "polyline": {
      "color": {
        "rgba": [
          0,0,0,255.0
        ]
      },
      "width": 3,
      "show":true
    },
    "vertexPositions":{
      "cartographicDegrees":[
          6.911114,52.253802,0,6.912797,52.267349,0,6.921599,52.275462,0,6.935299,
          52.280131,0,6.952367,52.282733,0
      ]
    }
  }
]
```

C.6 EXISTENTIAL ANIMATION WITH POLYGON VECTOR DATA

Listing C.6: CZML code for existential animation with polygon data.

```
[
    {
      "id":"document",
      "clock":{
        "interval":"2013-05-02T06:00:00/2013-05-06T06:00:00",
        "currentTime":"2013-05-02T06:00:00",
        "multiplier":34560.0,
        "range":"LOOP_STOP",
        "step":"SYSTEM_CLOCK_MULTIPLIER"
      }
    },
    {
      "id": "0",
      "availability": "2013-05-06T06:00:00/2013-05-06T06:00:00",
      "polyline": {
        "color": {
          "rgba": [
```

```
            0,0,0,100
          ]
        },
        "width": 3,
        "show":true
      },
      "vertexPositions":{
        "cartographicDegrees":[
           6.92065,52.268355,0,6.922077,52.270616,0,6.922732,52.269664,0,6.921126,
           52.267522,0,6.92065,52.268355,0
        ]
      },
      "polygon":{
        "show":true,
        "material":{
          "solidColor":{
            "color": {
              "rgba":[
                255,0,0,100
              ]
            }
          }
        }
      }
    },

    ...

     {
      "id": "8",
      "availability": "2013-05-05T18:00:00/2013-05-06T06:00:00",
      "polyline": {
        "color": {
          "rgba": [
            0,0,0,100
          ]
        },
        "width": 3,
        "show":true
      },
      "vertexPositions":{
        "cartographicDegrees":[
            6.917378,52.281145,0,6.922018,52.282572,0,6.932369,52.282335,0,6.937544,
            52.281799,0,6.942362,52.280907,0,6.94373,52.276802,0,6.943374,52.273293,
            0,6.946467,52.266868,0,6.952178,52.264905,0,6.950096,52.259432,0,6.941529,
            52.259789,0,6.938436,52.256339,0,6.934272,52.257588,0,6.936652,52.264013,
            0,6.92868,52.262823,0,6.928442,52.258123,0,6.921066,52.254614,0,6.913154,
            52.25854,0,6.909228,52.262585,0,6.907444,52.266333,0,6.906373,52.270378,0,
            6.909169,52.277338,0,6.917378,52.281145,0
        ]
      },
      "polygon":{
        "show":true,
        "material":{
          "solidColor":{
```

```
            "color": {
              "rgba":[
                 255,0,0,100
              ]
            }
          }
        }
      }
    }
]
```

C.7 STEPWISE ANIMATION WITH POLYGON VECTOR DATA

Listing C.7: CZML code for stepwise animation with polygon data.

```
[
    {
      "id":"document",
      "clock":{
        "interval":"2013-05-02T06:00:00/2013-05-06T06:00:00",
        "currentTime":"2013-05-02T06:00:00",
        "multiplier":34560.0,
        "range":"LOOP_STOP",
        "step":"SYSTEM_CLOCK_MULTIPLIER"
      }
    },
   {
    "id": "0-1",
    "availability": "2013-05-02T06:00:00/2013-05-02T18:00:00",
    "polyline": {
      "color": {
        "rgba": [
          0,0,0,255
        ]
      },
      "width": 3,
      "show":true
    },
    "vertexPositions":{
      "cartographicDegrees":[
          6.92065,52.268355,0,6.922077,52.270616,0,6.922732,52.269664,0,6.921126,
          52.267522,0,6.92065,52.268355,0
      ]
    },
    "polygon":{
      "show":true,
      "material":{
        "solidColor":{
          "color": {
            "rgba":[
               255,0,0,255
            ]
          }
```

```
        }
      }
    }
  },

  ...

  {
    "id": "0-8",
    "availability": "2013-05-05T18:00:00/2013-05-06T06:00:00",
    "polyline": {
      "color": {
        "rgba": [
          0,0,0,255
        ]
      },
      "width": 3,
      "show":true
    },
    "vertexPositions":{
      "cartographicDegrees":[
          6.917378,52.281145,0,6.922018,52.282572,0,6.932369,52.282335,0,6.937544,
          52.281799,0,6.942362,52.280907,0,6.94373,52.276802,0,6.943374,52.273293,
          0,6.946467,52.266868,0,6.952178,52.264905,0,6.950096,52.259432,0,6.941529,
          52.259789,0,6.938436,52.256339,0,6.934272,52.257588,0,6.936652,52.264013,0,
          6.92868,52.262823,0,6.928442,52.258123,0,6.921066,52.254614,0,6.913154,
          52.25854,0,6.909228,52.262585,0,6.907444,52.266333,0,6.906373,52.270378,0,
          6.909169,52.277338,0,6.917378,52.281145,0
      ]
    },
    "polygon":{
      "show":true,
      "material":{
        "solidColor":{
          "color": {
            "rgba":[
               255,0,0,255
            ]
          }
        }
      }
    }
  }
]
```