GAIA: A Grammar for Animated Infographics

Category: Research

Paper Type: please specify

Abstract—Animated infographics are a potent storytelling tool in various fields, from business to science communication. However, their creation is often laborious, involving meticulous element selection, property adjustments, and timing management. Therefore, we propose GAIA, a declarative grammar designed to streamline animated infographic creation. With GAIA, users can craft animations expressively and intuitively, abstracting away the underlying implementation. Via an abstraction of animated targets, GAIA provides an approach to express, save and share infographic animations from expert designers and allows novices to reuse them, significantly speeding up the creation process. We implemented a componentized GAIA compiler prototype and built a demo system based on it with diverse examples to showcase its usability. In a user study involving non-expert participants, individuals with no prior animation experience successfully replicated real-world animations within just one hour of learning. GAIA has been applied in several practical projects, further advancing the field of animated infographic authoring.

Index Terms—Infographics, animation, declarative language

1 INTRODUCTION

Animated infographics is considered a powerful tool for storytelling. It has been widely used in multiple areas, such as business, publicity, science popularization and so on. Compared to static infographics, it can interpret data or ideas more vividly and grabs the audience's attention quickly, appreciated and shared by people for years [12, 15].

Unfortunately, the creation of animated infographics is tedious and time-consuming, especially for novice users [9, 23, 35]. It can take weeks to authorize graphical animation itself that lasts around one minute [3]. Creating animated inforgraphics from scratch involve two major steps: static inforgraphics designing and animation creating [25, 35]. Initially, designers need to create static infographics (like te rightmost figure in Figure 1(A)) using feature-rich software such as Adobe Illustrator [2] and export them as Scalable Vector Graphics (SVGs) or other kinds of files so that downstream tools can be employed for animation creation. Then, the interactive tools (*e.g.,* Adobe After Effects [1]) are used to add animations. However, they only provide low-level operations, like keyframes or interpolation, for motion authoring, which requires expertise and is time-consuming [25]. Some users might choose programming tools (*e.g.,* D3 [14]) because they can precisely control property changes and organize timelines through program flow. In recent years, declarative languages are preferred by designers for this task as they favor conciseness over expressiveness [31]. Canis [19] falls into this category, which provides a declarative language for creating animated charts. These techniques simplify the creation process but can only use a predefined library of simple animations usually, which limits their usage. Here, we introduce two examples to better illustrate the challenges of creating animated infographics with existing tools.

Figure 1(A) illustrates four keyframes and the timeline of an animated. The creation process is as follows: To implement the CutIn effect in A1, each element needs to be masked separately. In A1, the y coordinates of title texts should be set at the start/end keyframe to let them move out from the masks. Then following the narration, three similar animation groups are created at the corresponding timestamp (A2-4). In each group, the bar enters using a Wipe effect (the mask is also required) and texts of the axis and the bar enter using CutIn. The relative offsets, durations and easing functions for each animation are carefully set to produce a harmonious result.

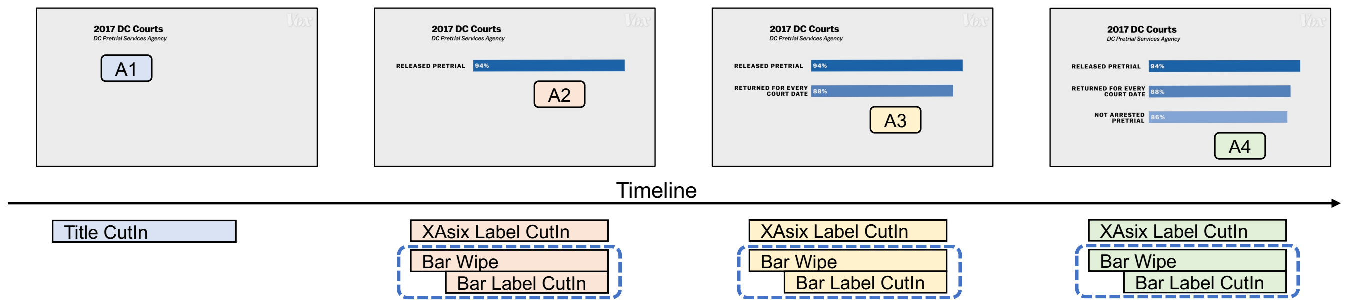
A more complex example is shown in Figure 1(B). Given a static SVG shown as the rightmost figure in Figure 1(B), five animations are added (B1-5). First, the chart framework enters at the beginning (B1) when the author explains this chart. Similarly, animations for each element should be handled one by one, using different effects depending on the element. For example, the lines use the WipeIn effect, while the x-label on the left uses the CutIn effect. Increasing offsets are set for the x-axis labels to achieve the stagger effect. Following the narration, the bar and bar label for the category "less than $20,000" enter (B2). A line and a label fade in to illustrate the average score (B3). Then, bars in the middle wipe in with a stagger (B4) to show a positive correlation. Finally, the corresponding bar and label for "More than $200,000" are animated with similar effects (B5).

Two obstacles hinder the designer's creation process. First, **non-intuitive and expert-driven creation approaches**. During the creation, each animation needs to: 1) select one animation target; 2) create a start keyframe at the appropriate timestamp and set initial properties; 3) create an end keyframe at the appropriate timestamp and set final properties; 4) specify an appropriate easing function. Designers need to deal with these cumbersome settings with an understanding of underlying concepts like keyframes, channels, *etc.* [28, 32] This is labor-intensive but unavoidable even for experts [25]. Second, **the repetitive creation of animations**. We observe that animations share similarities, not only between different element groups of the same instance but also across multiple similar instances. For example, animations framed in blue in Figure 1 are similar and applied on the bars and bar labels, which can be extracted and reused. As another example, entering animations of axes exist in many animated chart videos. As mentioned by Thompson *et al.* [36], reusing animations on similar groups is useful, which simplifies and speeds up the creation process. Given these situations, though these existing tools are helpful, designers of animated infographics still take a procedural approach to create them from scratch: draw the elements, select targets and create animations one by one.

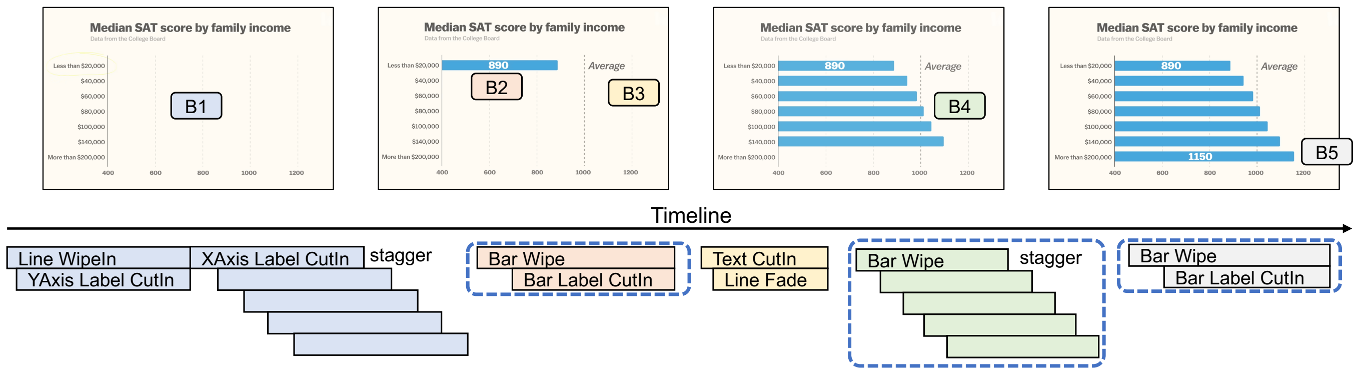
In this work, we aim to reduce the burden of creating animated infographics greatly. We propose GAIA, a declarative Grammar for Animated InfogrAphics. GAIA is motivated by three goals.

Expressivity: GAIA is both expressive and intuitive for reading and writing. In GAIA, animations are represented by effects applied to a group of visual elements instead of keyframes created individually for each separate element. Groups in one infographic (as we called, instance) form a hierarchy and each group can apply effects. Besides, grouping is also a way to organize animations [36]. Animations for groups can be combined with several alignment strategies (*e.g.,* start before previous), which also form a hierarchical structure simplifying the creation and refinement of timelines. GAIA provides a declarative grammar following these insights, which enables designers to create animations for groups and organize them in a hierarchical timeline. In this way, GAIA can support flexible and intuitive animation creation for infographics and enable users to express and refine complex animation timelines easily.

Reusability: Animations in GAIA can be reused. As mentioned above, reusing similar animations saves time and effort [24]. However, due to the diversity of animated objects, it is challenging even for a single simple animation, let alone a well-designed animation combo. In GAIA, animations are declared for a class of infographics, not a concrete instance. To this end, we conduct a target abstraction to



(A) Animated infographic example 1



(B) Animated infographic example 2

Fig. 1: (A) An animated infographic about the pretrial released defendants of the D.C. court in 2017 (retrieved from https://youtu.be/5mhwDSHKEyM?t=215). (B) An animated infographic about the relation between SAT score and family income (retrieved from https://youtu.be/WjVVwMGJ9S8?t=334). Similar animations are framed in blue.

describe graphical elements in infographics. The animations are not directly bound to the graphical elements but instead, bound to the role of the elements in a unified representation. For example, GAIA can declare a design for different types of animated elements in axes, such as domain, ticks and labels, without relying on a concrete instance. A wide range of SVGs from the internet or other design tools can be transformed into a format that contains the roles of each animated target and other information (like data). Then users can reuse complex but well-designed animations from themselves or other designers.

Extensibility: Template library in GAIA can be extended. Good design (including the coordination of effects, the design of parameters, *etc.*) is difficult to express, store, and share in multiple creations. Some existing declarative languages provide an animation library, but provided effects are preset and simple, like fade and wipe, designed for generic usage. Meanwhile, these techniques don't provide support for extending the animation library. The only way to do this is through a code-level implementation. Based on the target abstraction, GAIA allows expert designers to define new templates easily through a consistent declarative grammar as animation creation, then register them to the library and all users can use them just as other existing templates. For example, users can combine templates for bars and axes to create a new template for bar charts. Parameter abstraction is also introduced in GAIA so that a template can be more generic and customizable.

We implemented a GAIA compiler and released it as a TS library, which can be embedded in other tools as an animation engine with high-level specs as input. Through a demo system with a variety of examples, we show the expressive of GAIA. We conducted a user study with 8 participants with no animation design experience to illustrate the ease of use of GAIA and gather feedback. All participants can acquire the core features of GAIA within 1h of learning, and successfully replicate the data video designed by professionals. GAIA has been applied in a real-world project, and through interviews with three experts who employed it extensively, we garnered positive feedback and valuable suggestions for future improvements.

In brief, the main contributions of this work, in presentation order, include:

- **A set of high-level grammars** that enables intuitive creation of animations (section 3). Users can build complex animations via a hierarchical view and concatenating operations.
- **Target abstraction** decouples animation declarations from concrete instances, enabling reusability and extendability of GAIA, which is the first attempt for declarative animation grammars (section 4).
- **A compiler prototype**[1] and **workflow** of GAIA in different points of views (section 5).
- **An evaluation** on the usability of GAIA, including **a demo system** with a variety of examples used in the case study, a user study involved novice users, and semi-structured interviews conducted with experts experienced in using GAIA.

2 RELATED WORK

In this section, we discuss previous work related to GAIA from 3 aspects: infographic and animation for infographics, visual animation authoring tools and programming approaches for animation authoring.

2.1 Infographic and Animation for Infographics

The number of work on infographics published in recent years has increased. Infographics combine a variety of data-encoded elements and embellishments, which can attract readers' interests easily [10, 13, 21, 39]. Some existing researches attempt to understand infographics from semantic meanings of visual elements [30, 38]. Lu *et al.* [30] gave a concept of visual information flow. They pointed out one infographic

[1]The src code and the deployment version of both GAIA compiler and demo system are available in supplementary materials. An online version of the demo system is also available at https://gaia-ui.azurewebsites.net.

can contain multiple visual groups that share similarity. These groups are laid out to follow a backbone and form a hierarchy of infographics naturally. Wang's work [38] also illustrated this insight existed in a large range of real infographics. They named such groups as repeating units and pointed out that different visual elements in a group can have different roles semantically. Nested structure in the element hierarchy is common and can be easily created via some DSLs like D3 [14] and Protovis [22]. In the work of Liu *et al.* [29], visual elements are arranged by collections, which can be nested for organization. For example, some infographics contain several stacked bars, which form a multi-layered hierarchy: stacked bar groups and rectangles in each group. In GAIA, we follow the intuition of nested hierarchy in infographics to build an abstraction of animated targets.

Animated infographics are regarded as an efficient tool that can express an informative story using intuitive and attractive effects applied on visual elements [12,15]. Researchers have investigated a wide range of cases from the real world and conducted surveys for understanding and creation [35,38]. By collecting animations of infographics from multiple sources, like PowerPoint templates and video sites, Wang *et al.* [38] summarized the design space of animated infographics. They claimed that effects can be applied to a group of elements, then be organized via three arrangement styles. GAIA borrows the primitives of timing arrangement and adapts them to animations with a more complex timeline. Some other work focus on effects used in animations [35]. They derived animated visual narratives that fall into 8 categories of narrative strategies. These work inspire the design of animations in GAIA, but with the type system, GAIA can design animations for a semantic group of elements by combining these primitive effects.

2.2 Visual Animation Authoring Tools

Creation of animated infographics is a challenging task [3,24], so a few visual animation authoring tools are proposed. Keyframe-based tools, like Adobe After Effects [1] and Data Animator [37], provided low-level operations for motion creation. They allow users to specify the properties of visual elements on each keyframe and create intermediate frames according to the easing function. Besides, it is time-consuming when creating or modifying visual properties and time settings on a set of marks. GAIA doesn't need to define keyframes. Instead, high-level grammar is employed for easy reading and writing.

Template-based tools, as another category, provide a set of preset templates to facilitate animation creation. PPT [7], DataClips [9] and Flourish [4] fall into this category. In these tools, animations can be created intuitively and easily by applying templates on targeted elements instead of adjusting low-level properties. However, most of them only provide a small number of effects and can't be extended, which limits their applicability. Besides, their flexibility relies on pre-defined templates, thus are less expressive. As such, creating complex animations with them is difficult, even impossible. Data Player [34] interweaves data video authoring with narrations, providing a set of pre-designed animation presets. Katika [24] provides an end-to-end solution for creating animated graphics visually and also provides a motion library that can be extended by the community. Although it simplifies the creation pipeline, Katika can only employ statics graphics (called artworks in Katika) from its own embedded library. GAIA also uses a concept of templates and allows users to extend them smoothly. As a high-level grammar, GAIA and its compiler can be integrated as a part of interactive tools for animated infographics authoring. We refer interested readers to the survey on visualization authoring methods conducted by Grammel *et al.* [20].

2.3 Programming Approaches for Animation Authoring

To create animation more accurately and easily, some designers prefer programming approaches, which can be categorized into low- and high-level languages. Low-level languages, such as processing [8] and D3 [14], provide APIs for low-level operations that enable users to control the properties of animated targets directly. For example, D3 uses a transition operation that enables users to indicate what and how visual properties (*e.g.,* width and color) change on selected elements. Due to their high expressiveness, these languages have gained widespread use for creating custom and informative animations. However, they require users to have experience in graphic editions and animation creation, resulting in a steep learning curve. Users need to specify when and how these properties change for every element in detail, which is non-intuitive and time-consuming for complex animations with many animated elements.

High-level languages are typically preferred as they favor conciseness over expressiveness [31]. Some languages adopt JSON to express animations at a high level [18, 19, 19, 26, 27, 40]. Gemini and Gemini2 [26, 27] focus on transitions between two static visualizations. Animated Vega-Lite [40] is proposed for generating animations and binding logic for interactive graphics. With hiding under-layered mechanisms, some grammars describe animations as effects applied on selected elements, thus are more intuitive and concise compared to the low-level languages. For example, Canis and CAST [18, 19] provide several basic pre-defined effects to create animation without setting the initial and end values of properties. Via grouping operation, they can bind effects to a set of marks with a given align strategy thus the number of animations defined can be reduced. While these languages create satisfied animations quickly with concise specifications, they focus on data-driven charts instead of infographics. Another language, gganimate [5], uses concise code to express animations. Similar to Canis, it also employs encapsulation of animations to hide the details. However, it is mainly designed to create transitions for a specific chart type and can only be used in the programming language R.

Inspired by these work, GAIA is designed with a JSON-style grammar that strikes a good balance between expressiveness and conciseness. GAIA is the first one who considers types in animation authoring. According to our insights, elements in an infographic have different types and form a hierarchy. Exiting methods, like Canis, can only process groups of the same marks. Without awareness of types of animated targets, high-level languages can only provide generic and simple effects (such as fade and wipe in Canis, enter_fade in gganimate), which makes spec verbose when creating complex animations. Besides, pre-defined effect sets of existing tools are hard to extend. McNutt [31] pointed out the value of extensibility in DSLs, but none have provided a satisfactory solution for it in the field of animation creation. GAIA brings extensibility to this kind of grammar for the first time to create new animations or target types in a declarative way and enable reusing, which facilitates creation greatly.

3 THE GAIA GRAMMAR OF ANIMATION

GAIA is a high-level declarative language that enables rapid specification of animated infographics. Taking one static SVG as input, GAIA compiler generates a playable animated infographic instance according to the GAIA specification. In this section, we introduce the basic grammar to declare an animation in GAIA.

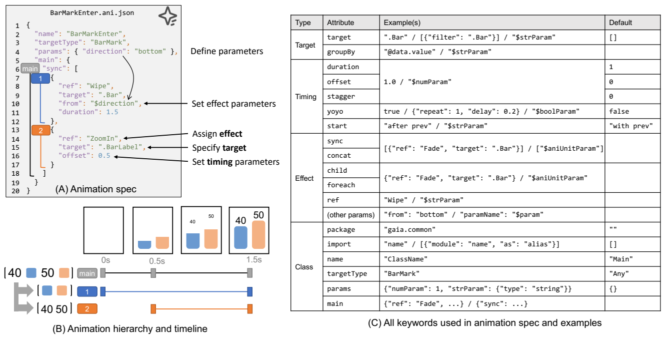
3.1 Animation Class and Unit

A GAIA animation spec (as we called *AniClass*) describes the animation applied on a given infographic with several animation units (as we called *AniUnit*) and a set of settings. Figure 2(A) shows an example of animation spec. It contains 3 *AniUnit*s: Sync animation, Wipe animation of bars, and ZoomIn animation of the bar labels. The animation created and the corresponding timeline are shown in Figure 2(B). To ease understanding, each attribute is associated with an attribute type given in Figure 2(C). Attribute `main` (line 5-19) defines the body of this animation, which contains a single *AniUnit*. Other attributes of type *Class* (line 2-4) address the reusability of GAIA, which will be discussed in subsection 4.4.

AniUnit describes the effects (using `ref`) applied on groups of elements (specifying by `target`). Formally, one *AniUnit* is an object that contains three types of attributes:

$$AniUnit := \langle Target, Timing, Effect \rangle$$

Target (subsection 3.2) declares the animated target. Attribute `target` defines a list of transforms specifying how an *AniUnit* picks and reshapes targets from its parent. Attribute `groupBy` can only be used


BarMarkEnter.ani.json

```
1  {
2    "name": "BarMarkEnter",
3    "targetType": "BarMark",
4    "params": { "direction": "bottom" },
5    "main": {
6      "sync": [
7        {
8          "ref": "Wipe",
9          "target": ".Bar",
10         "from": "$direction",
11         "duration": 1.5
12       },
13       {
14         "ref": "ZoomIn",
15         "target": ".BarLabel",
16         "offset": 0.5
17       }
18     ]
19   }
20 }
```

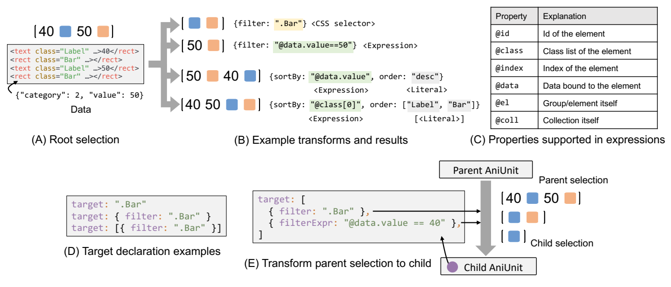
Type	Attribute	Example(s)	Default
Target	target	".Bar" / [{"filter": ".Bar"}] / "$strParam"	[]
	groupBy	"@data.value" / "$strParam"	
Timing	duration	1.0 / "$numParam"	1
	offset		0
	stagger		0
	yoyo	true / {"repeat": 1, "delay": 0.2} / "$boolParam"	false
	start	"after prev" / "$strParam"	"with prev"
Effect	sync	[{"ref": "Fade", "target": ".Bar"}] / ["$aniUnitParam"]	
	concat		
	child	{"ref": "Fade", "target": ".Bar"} / "$aniUnitParam"	
	foreach		
	ref	"Wipe" / "$strParam"	
	(other params)	"from": "bottom" / "paramName": "$param"	
Class	package	"gaia.common"	""
	import	"name" / [{"module": "name", "as": "alias"}]	[]
	name	"ClassName"	"Main"
	targetType	"BarMark"	"Any"
	params	{"numParam": 1, "strParam": {"type": "string"}}	{}
	main	{"ref": "Fade", ...} / {"sync": ...}	

Fig. 2: (A) An example of GAIA animation spec, *i.e., AniClass*, designed for BarMark. It contains 3 *AniUnit*s. (B) An animated instance and timeline of the animation created by this spec. *AniUnit*s form a tree hierarchy and each animation selects targets from the parent. (C) All keywords used in GAIA animation spec and examples.

with `foreach` (introduced in subsection 3.4), grouping the selected elements and applying the effect to each group. *Timing* (subsection 3.3) contains 4 attributes that describe the common timing settings of an animation. Attributes of type *Effect* (subsection 3.4) describe how to perform the animation. One *AniUnit* can refer to an existing effect using `ref` (like line 8, 14 in Figure 2(A)). Especially, *AniUnit* can be nested, using attributes `sync`/`concat`/`foreach`/`child` to combine child *AniUnit*s (like `sync` animation in Figure 2(A)). Other animation-related parameters also fall into this category. Figure 2(C) gives a full list of the attributes and provides some examples. Strings started with symbol $ are placeholders for parameters, which will be discussed in section 4.

3.2 Target and Target Selection

Each *AniUnit* has a *target selection*, which is a **list of elements**. It represents the elements that will be animated by this *AniUnit*. The root selection of an instance is the list containing all visual elements, where each element contains a set of key-value entries as the data (Figure 3(A)).


```
<text class="Label" …>40</rect>
<rect class="Bar" …></rect>
<text class="Label" …>50</rect>
<rect class="Bar" …></rect>
```

```
{"category": 2, "value": 50}
```

```
{filter: ".Bar"} <CSS selector>
{filter: "@data.value==50"} <Expression>
{sortBy: "@data.value", order: "desc"}
          <Expression>          <Literal>
{sortBy: "@class[0]", order: ["Label", "Bar"]}
          <Expression>          [<Literal>]
```

Property	Explanation
@id	Id of the element
@class	Class list of the element
@index	Index of the element
@data	Data bound to the element
@el	Group/element itself
@coll	Collection itself

```
target: ".Bar"
target: { filter: ".Bar" }
target: [{ filter: ".Bar" }]
```

```
target: [
  { filter: ".Bar" },
  { filterExpr: "@data.value == 40" },
]
```

40 50
Data
50
50 40
40 50
(A) Root selection
(B) Example transforms and results
(C) Properties supported in expressions
(D) Target declaration examples
(E) Transform parent selection to child
Parent AniUnit
Parent selection
Child selection
Child AniUnit

Fig. 3: (A) The root selection of the SVG animated in Figure 2. (B) Examples of transform operations and the corresponding results. (C) Properties that can be visited in the expressions. (D) Declaring transform operators in attribute `target`. (E) An example of transforming parent selection to child selection.

Attribute `target`, to be specific, declares how to generate target selection from its parent's selection via a sequence of *transform*s. Transform operators in the target attribute are concatenated, working like a pipeline between the parent and child *AniUnit*, allowing users to customize it according to their needs.

GAIA offers several basic transform operators, enabling the transformation of one selection into another:

- **Filtering** eliminates all unmatched elements in the collection using either a CSS selector or a predicate expression (two examples are shown in Figure 3(B)).
- **Sorting** reorders elements in the collection based on a key expression (two examples are shown in Figure 3(B)). The desired order can be specified using keywords `asc` or `desc` (only for numerical keys) or by providing a custom list.
- **Selecting** functions similarly to the select and selectAll operator in D3 [14], querying the descendants within one or multiple groups with a CSS selector. It allows GAIA to process <g> elements.

Other valid transforms include **Slicing** (extracting a portion by specifying a start, end, and step) and **Picking** (selecting one element using index). The expression will be evaluated during the compilation. When using expressions, GAIA allows users to visit several properties of elements via alias started with symbol `@`. Figure 3(C) gives a full list of supported properties.

3.3 Timing

As listed in Figure 2(C), Timing encompasses 5 attributes: duration, offset, yoyo, stagger, and start. These attributes govern the timing settings of an *AniUnit*. Offset is used by the parent animation (*e.g.,* sync or concat) to introduce a time delay. Yoyo enables the implementation of emphasizing animations (*e.g.,* blink, highlight, *etc.*) by specifying the number of repeats and the delay between each repetition. Stagger creates animations for each element in the list with an offset compared to the previous one, significantly simplifying "domino effects" as highlighted in [35]. It can be passed to some basic effect (*e.g.,* Wipe) or used with `foreach`. Similar to Canis [19], `start` control the reference for staggering in `foreach` animation, which can be `"with prev"` (start with previous) or `"after prev"` (start after previous).

One key consideration for GAIA, unlike other techniques, is the application of timing settings (*i.e.,* duration) within nested *AniUnit*s. In GAIA, each animation has 4 timing settings: (1) inheritance from its parent (*e.g.,* a sync animation with a specified duration), (2) its own individual settings, (3) settings computed from its children (*e.g.,* a sync animation consisting of several child animations with set durations) and (4) default settings. GAIA prioritizes non-null timing settings in order, and in case of conflicts, it rescales current animations or their children to resolve the conflicts.

3.4 Effect and Parameters

Attributes of type *Effect* include all effect-related settings. In fact, all animation effects can be specified by an animation type identifier along with a set of animation parameters. For instance, in Figure 2(A), the Wipe animation (lines 7-12) is invoked using the value of `ref` with `from` serving as the parameter passed to it. The parameters can be strings, numbers, boolean, and *AniUnit*s, which provides a flexible way to customize the animation. For string or number params, users can also use them as a part of other string values, which is useful when defining CSS selectors or expressions to process targets. As we mentioned above, params can also be used as local variables to prevent declaring several same pieces of spec. These *AniUnit*s, which include a `ref` field, act as the leaf nodes in the animation spec. GAIA offers a library containing 20+ common animations for different types of elements, such as Fade, Wheel, PathDraw, and more. What's more, users can register their own animations in the library and reuse them as needed. The library and reusability are the core of GAIA, as elaborated further in section 4.

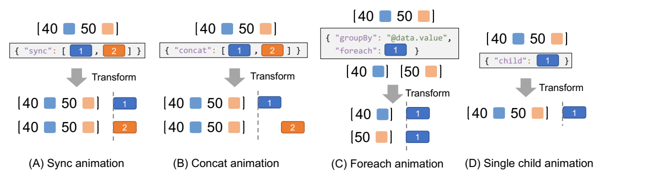


Fig. 4: Examples for non-leaf *AniUnit*s (sync, concat, foreach, single-child) in GAIA.

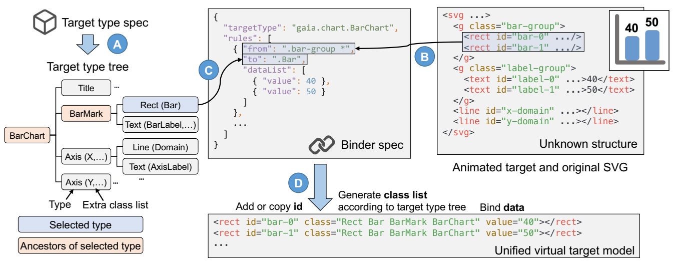
For non-leaf *AniUnit*s, *e.g.,* sync or concat, the children can be treated as one of the parameters as well. Figure 4 shows such *AniUnit*s in GAIA. In sync animation, all children are aligned to the start time of this animation. Concat animation, on the other hand, provides an alternative for building animation hierarchies, where the start time of each child is the end time of the previous one. In both cases, the `"offset"` value defined in the children can further refine the declaration. In the foreach animation, the elements in the parent selection are first grouped by the value computed by `"groupBy"`, and then GAIA creates a child animation for each group. The single-child animation will simply apply the effect on its selection. It is useful when the inner animation is defined as a parameter. Except for the foreach animation, all other animations pass the parent selection to the child animation directly with transforming.

4 REUSABILITY AND EXTENSIBILITY

Some infographics have similar semantic components, and animations of these components are also similar. Reusing these animation benefits creating progress [36]. However, their structures in the form of SVG differ from each other. For example, the order of elements or hierarchy made by <g> might vary inside SVG code. Reusing animation across different instances requires a unified target abstraction. In the previous section, we introduce the attribute `target` in animation spec, discussing how GAIA manages the targeted elements of each *AniUnit*. In fact, these target specs are performed on an abstraction layer, as we call *virtual target model*, instead of original SVGs. In this section, we will discuss the target abstraction and explain how it enables reusability and extensibility of GAIA animation spec.

4.1 Target Abstraction and Reusing

The virtual target model employs the same target selection mechanism in subsection 3.2. However, in the virtual target model, elements in the list of target selection are not the original SVG elements. To provide sufficient abstraction capability, the design of the virtual target model should be carefully considered. We found that the animated targets have a tree hierarchy, which is similar to the SVG DOM tree or the concept of scene graph [33]. According to the insights from Kim *et al.* [26], users might specific sub-elements like *the x-axis title*. Motivated by these findings, we design the target abstraction as a tree-like structure to support such expressions. Specifically, the target abstraction is a tree shown on the left side of Figure 5, as we call *target type tree*.



```
{
  "targetType": "gaia.chart.BarChart",
  "rules": [
    { "from": ".bar-group *",
      "to": ".Bar",
      "dataList": [
        { "value": 40 },
        { "value": 50 }
      ]
    },
    ...
  ]
}
```

```
<svg ...>
  <g class="bar-group">
    <rect id="bar-0" .../>
    <rect id="bar-1" .../>
  </g>
  <g class="label-group">
    <text id="label-0" ...>40</text>
    <text id="label-1" ...>50</text>
  </g>
  <line id="x-domain" ...></line>
  <line id="y-domain" ...></line>
</svg>
```

```
<rect id="bar-0" class="Rect Bar BarMark BarChart" value="40"></rect>
<rect id="bar-1" class="Rect Bar BarMark BarChart" value="50"></rect>
...
```

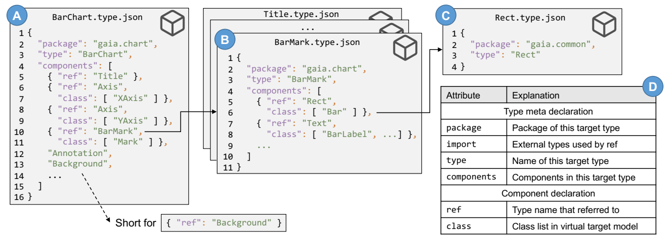
Fig. 5: Steps to create a virtual target model, which can be used as a unified representation of a class of infographics.

Target type trees are defined using a JSON-style spec, named target type spec (Figure 5(A), introduced in subsection 4.2). Then, with a binder spec (subsection 4.3), the original SVG elements can be bound to a semantic type by linking it to a node of the target type tree (Figure 5(B, C)). Then a virtual target model can be generated (Figure 5(D)), which is a list of virtual SVG elements.

With the virtual target model, all animated instances, *i.e.*, SVG, of the same type can be represented as a list of elements with class lists defined by the target type tree. The target attributes in animation spec are applied on this virtual layer, which enables the reusability of animation spec regardless of the structure of SVGs. The expert designers can represent and share their animation designs as *AniClass*es and end users can simply reuse them using `ref` keyword.

4.2 Target Type Spec

The target type spec, which defines the target types and their components, is also in JSON style. Inspired by the object declaration in JSON schema [6], the target type spec can declare other target types as its components via `ref` attribute. As shown in Figure 6(A-C), the type declaration of BarChart contains components defined in other target type spec, declared a target type tree for BarChart shown in Figure 5(A). All these specs can be reused to form other target types, such as LineChart or Timeline. GAIA manages these target types via packages and provides a set of target types for common infographics and their components.



```
{
  "package": "gaia.chart",
  "type": "BarChart",
  "components": [
    { "ref": "Title" },
    { "ref": "Axis",
      "class": [ "XAxis" ] },
    { "ref": "Axis",
      "class": [ "YAxis" ] },
    { "ref": "BarMark",
      "class": [ "Mark" ] },
    "Annotation",
    "Background",
    ...
  ]
}
```

```
{ "ref": "Background" }
```

```
{
  "package": "gaia.chart",
  "type": "BarMark",
  "components": [
    { "ref": "Rect",
      "class": [ "Bar" ] },
    { "ref": "Text",
      "class": [ "BarLabel", ...] },
    ...
  ]
}
```

```
{
  "package": "gaia.common",
  "type": "Rect"
}
```

Attribute	Explanation
Type meta declaration	
`package`	Package of this target type
`import`	External types used by ref
`type`	Name of this target type
`components`	Components in this target type
Component declaration	
`ref`	Type name that referred to
`class`	Class list in virtual target model

Fig. 6: Target type can be declared using target type specs (A-C) with a specified type name and components. (D) lists the attributes used in target type spec.

Figure 6(D) gives a detailed explanation of attributes in target type spec. Notice that `package` can be used to avoid name conflicts. In this case, these specs are in the same package so no import is needed (common types don't need to import). The name of the type and class list declared in components will be used in the class list of nodes in the target type tree.

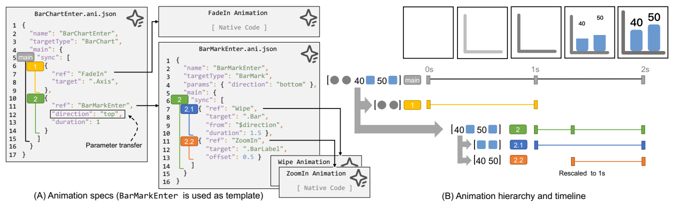
4.3 Binder Spec

Binder spec is used to bind the original SVG elements to the target type tree, and then generate the virtual target model. As shown in Figure 5, a binder spec contains attributes `targetType` and `rules`. The value of `targetType` determines the target type tree to use. Each rule in the `rules` (1) selects a set of elements in the original SVG by a CSS selector in `from`, (2) selects exactly one node in the target type tree by a CSS selector in `to`, and (3) binds the data to each element one by one according to the `dataList` attribute. Element id can also be set if needed. The result is called virtual target model, which is a list of transformed SVG elements. Each element has (1) id (GAIA will generate one automatically if no id is available), (2) a list of class names that come from the selected type node and its ancestors, and (3) data written to the attributes of the element. This list will be the root selection of the animation spec, and the `target` of the *AniUnit* at `main` will be applied to it.

4.4 Using Animation Spec as Template

One *AniClass*, introduced in subsection 3.1, can be directly stored as a template and reused in other animation specs or further be a part of another template. Notice that *AniClass* can be declared without any concrete instance because the target is declared according to the virtual target model. Now taking the animation in Figure 7 as an example, we discuss how *AniClass*es in GAIA cooperate to represent a complex animation. Notice that end users, only need to specify the top *AniClass* for customizability, where they can freely reuse animations from expert designers, so the progress is accelerated greatly.

Figure 7(A) shows an *AniClass* designed for BarChart. It refers to two sub-*AniClass*es, FadeIn on axis components and BarMarkEnter (as same as the spec in Figure 2(A)). The BarMarkEnter further refers to other *AniClass*es stored in the library, though form an animation hierarchy. In this case, the animation spec BarMarkEnter declared using JSON is composited as a sub-animation. It picks targets from the outer scope and the timing settings are dominated as well (rescaling duration if needed). Parameters set in the outer scope will be transferred



BarChartEnter.ani.json
```
1  {
2    "name": "BarChartEnter",
3    "targetType": "BarChart",
4    "main": {
5      "sync": [
6        {
7          "ref": "FadeIn"
8          "target": ".Axis",
9        },
10       {
11         "ref": "BarMarkEnter",
12         "direction": "top",
13         "duration": 1
14       }
15     ]
16   }
17 }
```

FadeIn Animation
```
[ Native Code ]
```

BarMarkEnter.ani.json
```
1  {
2    "name": "BarMarkEnter",
3    "targetType": "BarMark",
4    "params": { "direction": "bottom" },
5    "main": {
6      "sync": [
7        { "ref": "Wipe",
8          "target": ".Bar",
9          "from": "$direction",
10         "duration": 1.5 },
11       { "ref": "ZoomIn",
12         "target": ".BarLabel",
13         "offset": 0.5 }
14     ]
15   }
16 }
```

Wipe Animation

ZoomIn Animation
```
[ Native Code ]
```

main
1
2
2.1
2.2
Parameter transfer
(A) Animation specs (BarMarkEnter is used as template)
40
50
0s
1s
2s
Rescaled to 1s
(B) Animation hierarchy and timeline

Fig. 7: An example of using animation spec as a template to declare and instantiate a complex animation.

to the sub-animation. Applying an SVG example, the final timeline and selected elements for each declared *AniUnit*s are shown in Figure 7(B).

Attributes of *Class* specify more information of one *AniClass*. Attribute `import` specifies other *AniClass*s used by `ref` keyword, which prevents name conflicts (common *AniUnit*s don't need to import). *AniClass* `name` is used when other *AniClass*s refer to and reuse it (`"Main"` as default). The target type that this *AniClass* associates with is declared by `targetType`. Attribute `params` allow users to set the parameters passed in when reusing, leaving the customizability of *AniClass*s. An example is shown in line 12 in Figure 7(A).

We want to emphasize that this mechanism does not rely on any concrete infographic instance. For example, in this case, users create an *AniClass* for BarChart via combining existing designs for components, *i.e.,* axes and marks, without referring to a given bar chart SVG. Then this *AniClass* can also be registered to the library and used later. All these things can be done in, but not limited to, a JSON-style spec, to implement the extensibility of GAIA. Designers can share their animation designs with other users for reusing without the involvement of GAIA's developers.

5 WORKFLOW AND GAIA COMPILER

In this section, we first put things together, introducing the workflow of GAIA, then discuss the implementation of a highly modular compiler of GAIA.

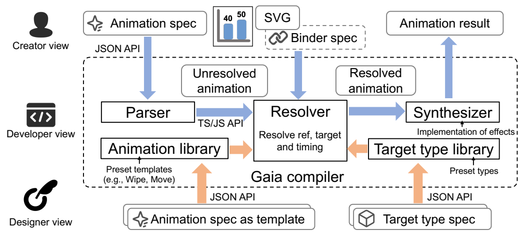


Fig. 8: Workflow of GAIA from three views and implementation of GAIA compiler.

5.1 Workflow of GAIA

As shown in Figure 8, the workflow of GAIA has three views.

1. **Creator view**. For end users whose expectation is a visible animation, GAIA hides the most complexity of designing detailed animations and implementing effects. They only need to specify their top-level animation spec and provide an infographic with declared types. Creators can be human users or tools that use GAIA to create animations.

2. **Designer view**. Animation design experts can design target types and provide high-quality animation templates for them. They don't need to care about the structure of the real infographic, nor be aware of the implementation. Instead, they simply focus on animations from themselves or other designers using high-level grammar, creating animations that are consistent in style and mood, and clear in meaning.

3. **Developer view**. Developers need to implement basic animation effects and serve them as code-native *AniClass*es. In our implementation of GAIA's compiler, this work is minimized as the modular design (subsection 5.2). Most of the GAIA compiler components don't require any change when adding effects.

The border between creators, designers and developers is not clear-cut. Creators can also build their templates and types for convenience. Developers can also define *AniClass*es, like designers, with TS/JS API. Anyway, Gaia decouples the final creator of the animation, the professional designer of the animation template and the developer of the animation effect from each other, so that each part can be developed and changed separately, and the complexity of other domains can be hidden. This stems from Gaia's unique reusability and extensibility.

5.2 GAIA Compiler Prototype

GAIA frameworks need to provide consistent, efficient APIs for different users, while ensuring reusability and extensibility at the framework level. We implemented a prototype of GAIA compiler and published it as a library. The middle part of Figure 8 illustrates the pipeline of the compiler. For the creator side, the compiler takes the animation spec and the infographic as input and outputs the animation result. The pipeline involves three steps: **parsing**, **resolving** and **synthesizing**. First, for any high-level spec written in JSON, a parser is used to transform it into a GAIA internal object using TS/JS API. The animation spec, in this step, will form a data structure called **unresolved animation**, which is a tree-like structure. Next, a virtual target model is created with the declared types and the given SVG. Then, the compiler resolves the animation, linking the animations declared with `ref` from the library, resolving selected targets on the virtual model, and computing duration. So far, this **resolved animation** does not rely on any specific implementation details but contains all the information for the animation. It can be returned to the user layer for preview, debugging, or building UI (if the creator is a tool using GAIA). Finally, the compiler synthesizes the resolved animation and creates the animated instance with the implementation of effects (which are tackled by developers).

It is important to note that GAIA provides a complete end-to-end system for creators and designers, but each component used in the compiler pipeline can be replaced. For instance, an alternate parser can be supplied so that new features of JSON spec, or other high-level grammars, can be hired to construct unresolved animation objects. As another example, the synthesizer can have a different implementation of the same *AniClass* to support different element types, such as HTML and SVG, in a seamless way for high-level spec users. Besides, the animation and target type library can be remotely stored and fetched. We implemented a RESTful server as a remote library for internal use, which allows users to upload and download animation specs and target type specs.

6 EVALUATION

In this section, we first illustrate the expressivity and reusability of GAIA by an animation gallery with a wide range of examples[2]. We conducted a user study with non-experts in animation creation, where we focused on observing the ease of use and the learning curve of GAIA. Finally, we performed semi-structured interviews with the experts who have used GAIA in their projects, to further understand the advantages and limitations of GAIA.

6.1 Example Gallery

GAIA's expressiveness lies in two aspects: the compatibility of different kinds of infographics created by different tools and the capability to efficiently express a variety of complex animations. Therefore, we built an animation gallery with a wide range of examples including (1) replications of real-world data videos with different topics and styles; (2) generated visualizations created by Vega-Lite; and (3) animations of infographics adapted from related work [17, 19, 38]. These examples not only cover a wide range of infographic types, such as isotypes, timelines and different kinds of informative charts but also involve all

[2]Available in supplementary materials

animation semantic types introduced in [16]. Each example includes a top-level animation spec, a set of templates used (some of them are reused from other examples), a static SVG and a corresponding binder that maps the SVG to the target type tree[3].

As a key design goal of GAIA, reusability of GAIA enables sharing the animation design. A lot of reuse in our gallery also simplifies the complexity of specs. For instance, template `BarChartEnter` is used in the example *Australia Fire*, *Transportation Mode* and *Wealth Inequality*. In the example *GDP Comparison* and *2017 DC Courst*, template `ShowCategory` are invoked with different parameters, as they all include several similar entering animations for bars, bar labels, and axis labels/symbols. To further show the reusability, we use templates to create all animations in the examples for Vega-Lite generated charts. The template `ChartFrameworkEnter` is reused across the bar chart, line chart and scatter plot. We also design generic templates used to animate series data for each different chart type with name `StaggeredXXXSeriesEnter`. They animate different types of marks in a staggered way and can be reused in other charts with similar mark types. All these templates leave the flexibility for users to customize the animation by adjusting the parameters.

Note that we also create an animation with a reusable template for a Vega-Lite bar chart without a binder, as such SVGs are generated following the same structure. The chart is bound to the type `Any` so the original SVG structure can be used directly. This case demonstrates the case without the binder, where GAIA can still offer the necessary support to create animations flexibly. This feature is suggested by one of the users of GAIA (the first interviewee in subsection 6.3), which considers the case when users don't need the reusability supplied by the binder and virtual target model.

6.2 User Study

In our user study, our objective is to assess the usability of GAIA among novices with diverse backgrounds. The central focus of our study revolves around the replication tasks. Because GAIA, as a descriptive language instead of an interactive creation tool, may present challenges to beginners. We hypothesize that even novices with no prior knowledge can use designer templates to replicate real data videos after limited learning, which is sufficient to illustrate the user-friendliness and rationality of designs like templates.

6.2.1 Study Design

We invited 8 participants (denoted as P1-8, aged from 22 to 28) to our user study. All participants possess backgrounds in computer science, with P1 through P5 specializing in visualization and P2 having a background in storytelling. Besides, P6 also has experience in visualization. None of them have prior experience in animation creation.

We ran the study through an online meeting one by one and each participant connected to the meeting via a desktop computer with mouse and keyboard. They were asked to share the screen and all the interactions were recorded. We prepared a video tutorial to introduce basic concepts about the creation of animated infographics. Then we asked participants to complete 4 replication tasks in a web-based system[4]. Before each task, we introduce the related features of GAIA, then provide an example to illustrate the task process. These examples use different infographics and animations, but are implemented in the same way as in the task.

Task 1 to 3 involve the replication of the animated infographic shown in Figure 1(a), while Task 4 entails replicating Figure 1(b). During each task, we answered questions from participants and provided hints when they were stuck, but we avoided providing a direct answer. We recorded the time spent on each task and the problems encountered by participants. The details of the tasks are as follows:

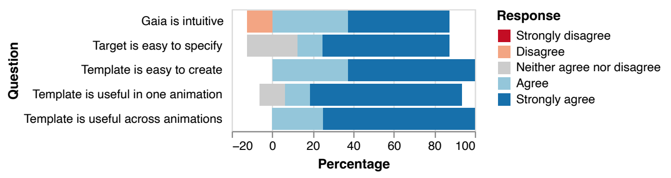


Fig. 9: User evaluations of primary features of GAIA using a 5-point Likert scale.

- **Task 1** Adjust the effect, stagger, and offset in a given animation spec to complete the replication. This task is to make sure participants understand the basic grammar of GAIA.
- **Task 2** Set the target and animate each element one by one to complete the replication. Participants can understand the target abstraction and experience creating animations without involving reuse.
- **Task 3** Create and use a template to complete the replication. We aim to let participants compare Task 2 and appreciate the benefits of reuse.
- **Task 4** Create a new animated infographic with the template created in Task 3 and other templates provided. This task is to let participants experience the whole process of creating animations with generic templates.

It takes about 60 minutes to watch the video and complete the task. After finishing the tasks, we asked participants to fill out a questionnaire to collect their feedback on the primary features of GAIA related to each task. We conducted a semi-structured interview as a follow-up to gain insights into their experience with GAIA and gather their suggestions.

6.2.2 Results

All participants could master the core features of GAIA in about 1 hour and use the learned abilities and provided templates to replicate the animation shown in Figure 1(b). The average time spent on each task is about 10.63 (standard deviation SD=4.14), 6.13 (SD=2.03), 4.50 (SD=0.93), and 11.38 (SD=3.70) minutes, respectively. The time spent on Task 1 is the largest among the first three tasks, which is mainly due to the unfamiliarity with the basic concepts of animation creation and the grammar of GAIA. Some participants (3/8) struggled with the concept of offset and stagger in Task 1 so they cost more time. After Task 2, participants are more familiar with GAIA, and the new concept – templates – is not expensive to learn, so Task 3 takes the least time and has a small standard deviation. Task 4 contains several complex animation groups, but all of them were able to replicate within 20 minutes.

Figure 9 shows the results of the questionnaire. The overall satisfaction of GAIA is high. Most of the participants (7/8) think GAIA is intuitive and easy to use (average AVG=4.25, SD=1.04, 1 means strongly disagree and 5 means strongly agree). P8, who gave a rank of 2, said, "*I couldn't understand the stagger at first, so I got stuck.*" However, he reported that after understanding these basic concepts, writing GAIA became easier. For the target selection, most participants (6/8) took a positive attitude (AVG=4.28, SD=0.92). The template, one of the most important features of GAIA, is considered to be easy to create (AVG=4.63, SD=0.52), and proves beneficial for crafting individual animated infographics (AVG=4.63, SD=0.74) as well as multiple animated infographics (AVG=4.75, SD=0.46).

In the interview, all participants agreed that GAIA is an easy-to-use and powerful tool for creating animated infographics. "*It let me pay more effort on design instead of effect implementation.*" P5 commented. Participants thought that certain key features, like target selection and templates, would not incur additional learning costs. "*People who have used D3 or DOM APIs can easily get started (specifying target)*" (P1). Besides, some participants think the target model helps simplify the creation. P6 told us: "*target selection is a list of elements, which is more clear compared to the DOM tree.*" Some participants mentioned that the logic of creating and using templates is similar to functions

[3]For SVGs generated by Vege-Lite, binder specs can be generated automatically because the structure of SVGs is fixed.

[4]The system is the same as the demo system for case study. Materials of the user study can also be found in the demo system.

(P2, P3, P5), and the parameters can be set intuitively (P6, P7). As for the benefits of the template, most of the participants agreed that it saves time and facilitates exploration. P4 said, "*It (template) avoids to adjust elements one by one when animations are similar but different in target/duration*". "*It is also helpful when we need to modify the animation since we only need to change the template*" (P3).

We paid attention to the mistakes participants made during the task, and tried to analyze the causes. Setting the wrong parameters or using the wrong template are the most common mistakes. For example, P5 tried to set the `from` parameters to `Fade` in Task 1. GAIA compiler ignored the wrong parameters without any warning, causing him to start doubting whether the code had compiled successfully. This problem was also mentioned by P1 and P7. "*The system will not highlight the modified. Sometimes I wonder if the changes have taken effect*" (P1). In addition, we also noticed that the wiping direction of the bars and lines in Task 4 is easily overlooked. After our hints, participants still had to switch and watch the video several times before they realized the difference.

We also asked about the disappointments and improvements for GAIA. More than half of the participants (P1, P2, P3, P5, P7) mentioned the lack of links between specs, SVG, and target models is a pity. This made the debugging difficult. "*It is tedious to find useful resources in different tabs. Develop an IDE for* GAIA *might be cool*" (P3). Besides, P3 and P6 suggested that if lambda expressions can be used to compute some values (like duration or stagger), it will be more flexible. Three participants (P2, P3, P6) mentioned there is no gallery or document to support exploration, which is useful for novices.

6.3 Expert Interview

GAIA has already been used in real-world practices by several experts in the domain of animation creation and visualization. We kept in contact with these experts and collected their feedback during the development of GAIA. In this section, we interviewed three of them (denoted as E1-3) in a semi-structured way to gather their perspectives from different views (creators and designers). E1 has participated in some projects related to animation creation and has expertise in animation design. E2 is a researcher in visualization and data analysis, having experience with animation creation for more than 3 years. The research direction of E3 is data storytelling. She has experience using animation creation tools for more than 6 years.

6.3.1 Existing and Potential Usage Scenarios

E1 used GAIA, from the view of a creator, to create videos for two months up to the time of the interview. Both E2 and E3 use GAIA as the underlying animation engine to complete research work on interactive authoring and automatic generation of data videos. They used GAIA for two years and one year, respectively, gaining insights into its functionality from the perspectives of creators and designers. The work on E2 is being delivered, and the work of E3 has been published in the top conferences of visualization. All of them think GAIA reduces the complexity of their tasks, simplifying the workflow and avoiding code-level implementation of motion effects. E1 said, "GAIA *is more powerful and customizable than other tools like PPT and can be easily integrated into our project, linking other code to the animation engine*". E2 and E3 tried to use GAIA as an abstraction layer for incorporating large language models (LLMs) into the process of generating data videos. "*The most valuable aspect is that it is an abstraction layer to express, save, and reuse animations*", E2 told us.

When discussing the potential usage of GAIA, E1 commented that GAIA can support animation design in larger scenarios, not limited to 2D graphics (*e.g.,* SVG), such as VR and 3D. Besides, E1 mentioned that GAIA can contribute to the adaptive UI design, such as the transition between two UI layouts. Both E2 and E3 think that GAIA can benefit a wide range of users, including researchers, designers and data scientists. "*As a researcher on storytelling, I am one of the beneficiaries. In addition,* GAIA *can also help popular science video producers save time*", E3 said.

6.3.2 Expressiveness and Learning Curve

All of them agree that GAIA spec is simple and well structured, having a good balance between expressiveness and conciseness. "*All animations I need can be done via changing the parameters of templates*", E3 said. The JSON grammar and CSS selectors make GAIA easy to learn and use, especially for users with a programming background (E1). "*The animations I designed are usually complex.* GAIA*'s grammar offers a controllable and predictable way to specify them.*", E1 said. Compared to other tools like D3 and animated Vega-Lite, both E2 and E3 think GAIA targets a different problem and is more suitable for creating animated infographics, as other tools focus on transition and interaction, or employ a data-driven approach.

However, E1 pointed out that it is difficult for GAIA to express some relatively small animations, such as the blinking effect of a cat. *"My project manager thinks a large movement or stagger of multiple elements is more difficult to achieve than a small effect. But (in* GAIA*) the opposite is true" (E1).* E2 mentioned that when integrating GAIA into an automatic workflow, the parameters make things difficult. "*Some highly stylized or unusual animations are difficult to create in* GAIA", said E3, "*they might require familiarity with* GAIA *or implementation through code.*" Overall, though, all agreed that the ratio between the capabilities offered by GAIA and the cost of learning was high.

For the learning curve of GAIA, E1 and E3 believe that users still need to understand the JSON format and some basic knowledge in visualization and animation creation, but after that, the curve becomes very flat. "I tried to explain my animation design to a project manager using GAIA, but I find it is difficult as they even don't know the JSON" (E1). "*The difficulty of getting started depends on other experience with similar specs and background knowledge*", E3 said, "*but for a programmer, this is not a problem.*" These insights are also consistent with what we observed in the user study. E2 told us, "*it is much easier than D3, and has a similar learning curve to animated Vega-Lite.*"

6.3.3 Template Usage

GAIA is the first high-level grammar that supports the template and supplies reusing for animations. E1 thought the template is like functions in programming languages and can be used to encapsulate animations. "*I use templates all the time*", E1 told us, "*It allows me to manage animations more clearly, even sometimes I don't reuse them.*" In the work of E2, some complex animations are encapsulated as templates with readable names describing the effect (*e.g.,* `Pie-wheel-in-and-legend-fly-in`). Then providing these animation names, element information (obtained from the target model) and narrations as prompts, LLMs can generate animations automatically. "*LLMs can understand the meaning of the animation, combine the narrative with the appropriate animation for the elements without considering the implementation details*", says E2. E3 said that as long as a template meets the needs, she will use it first. "*They are intuitive and save time*" (E3).

6.3.4 Extensibility

GAIA allows users to create templates and extend the library in a declarative way. E2 and E3 collaborated for an extended period. As designers, they all craft templates according to their own needs. E2 mentioned, "*Sometimes we share the templates we have created and leave the ones we might use in our projects.*" E2 created more than 20 templates using high-level specs, but E3 doesn't think it is necessary to create so many templates. "*Templates are most commonly used animations. A few templates will cover the needs of most cases*" (E3).

As for the consideration when designing templates, E1 thought the key point was the complexity. "*I hope the template is sufficiently generic without being overly complex to comprehend, as this would be detrimental for both myself and my collaborators*" (E1). E2 commented the design of templates should consider the style of animated elements. He also pointed out that some settings in a template are more likely to be changed to fit different scenarios. As an example, he mentioned that bars might enter quickly at the beginning in some cases, but slowly in others, so letting the duration of these key elements be customizable is a good idea.

E2 also provides some insights for GAIA improvements. He suggested implementing effects for specific purposes. For example, the effects used to create animations for children need to be more fun (like bar bounce). In addition, emotional factors may also be a consideration to expand the effects.

7 DISCUSSION

GAIA is the first declarative grammar for animated infographics, which provides capabilities of flexible expression and reusability. In this section, we discuss the design considerations, possible usages and limitations of GAIA.

7.1 Cognitive Dimension

The design of GAIA takes into account the Cognitive Dimension [11]. First, a user might act in different roles when using GAIA (as we mentioned in section 5), so the consistency of the grammar is a concern. GAIA uses the same grammar for both animation and template. Consistent with the findings in the user study, this significantly reduces the cost of creating and using a template. Besides, the tree-like spec logic for animations and target types is similar. Keeping some attributes consistent, such as ref, also promotes understanding. Second, the nesting animation of GAIA provides a progressive learning style and supports *AniUnit* level evaluation. For *AniClass* authoring, users can design a simple animation first, and then gradually add more details, each time the *AniUnit* can be evaluated immediately. In user studies, most participants use this approach, reviewing the results as they make changes. Third, users have the freedom to control their work steps. For a complex animation, users can break it down into multiple specs (*AniClass*). Each spec can be evaluated individually and referred by a top-level animation so that the entire project can be completed incrementally, as stated in E1's feedback.

7.2 Trade-off of Target Abstraction

GAIA decouples the infographics creating and animation designing and employs a virtual layer for reusability. However, users need to clarify the target type and understand the virtual model before designing animations, which might require more effort in some cases. We determined that although the virtual layer introduces extra learning costs and usage burdens, these are outweighed by the convenience of reusing templates from professional designers, which is a relatively harder part compared to declaring the types. In addition, virtual target models bring expert-designed animations from one instance to a class of instances, helping to express, encapsulate, and transfer domain knowledge, creating a more prosperous community environment. As another insight, the design of the virtual target model (*e.g.,* list in GAIA) can be used to eliminate the complexity of real structures (as P6 said). Finally, in the case study, we also show that GAIA can fall back to the original SVG structure when the virtual layer is not needed.

7.3 Potential Usages

GAIA has the potential to be used in many more scenarios. GAIA is a high-level spec that can be easily read or written directly, but it can also be used as a part of interactive tools for animated infographics authoring (E2, E3). Benefiting from the reusability of GAIA, beginners will be able to design professional animations faster and their designs can also contribute to the tools' ecosystems. Even without exposing the reuse concept to the user, tools can take advantage of GAIA's expressiveness and animation library to get the job done. Furthermore, GAIA offers a well-organized format for representing both infographic instances and animations, which can obtain benefits from generative AI models. For instance, tools can incorporate AI to aid in animation creation with GAIA as an abstraction layer (E2), swiftly beginning with blank projects or carrying out design completion. It can even be employed to construct end-to-end generation tools for animated infographics, which is our ultimate objective and further work. For now, the prototype of GAIA has already been used in some research projects for animated infographics authoring and attracted some enterprise engineers for further cooperation.

7.4 Debugging

Debugging declarative grammar is challenging [33], and GAIA is no exception. This issue is also mentioned by some participants in the user study. With our experience working with downstream users, problems (*e.g.,* wrong syntax, empty selection and wrong parameters), will consume a lot of energy. We implemented GAIA with these issues in mind, providing a complete logging system and error-handling flow. In addition, information related to the animation logic (such as selected elements, final animation structure, computed durations of each animation) can be obtained by the resolved animation object to facilitate debugging. It can also be used by developers of upper-level tools for visual animation and debugging. Even so, we found that new users of GAIA still often ran into problems and struggled to get effective tips for changes. Smoothing the development pipeline and debugging approaches for GAIA is a direction worthy of further study.

7.5 Limitation and New Opportunities

GAIA introduces the target abstraction, then decouples the work for creators, designers and developers. However, this workflow has not been fully validated. The evaluation of expressivity is also demonstrated through the repetition of examples, rather than through the animation result implemented by professional designers. In addition, we did not go into depth and validate the design of the target type spec and binder spec. P5 mentioned the generalization of target types and corresponding templates can be further explored. In expert interviews, E2 shared his experience in creating generic templates, but further research was lacking. In addition, P5 also asked whether binder specs can be generated automatically or type binding can be done interactively. These are the directions that can be further studied in the future.

8 CONCLUSION

In this paper, we present GAIA, a declarative animation grammar for animated infographics. GAIA is designed to be expressive, reusable, and extensible. We hope that GAIA will promote relevant academic research and practical commercial use of animated infographics.

REFERENCES

[1] Adobe after effects, 2023. 1, 3
[2] Adobe illustrator, 2023. 1
[3] The animation process – complete step by step guide, 2023. 1, 3
[4] Flourish | data visualization & storytelling, 2023. 3
[5] A grammar of animated graphics | gganimate, 2023. 3
[6] Json schema. `https://json-schema.org`, 2023. 5
[7] Microsoft powerpoint slide presentation software, 2023. 3
[8] Processing.org, 2023. 3
[9] F. Amini, N. H. Riche, B. Lee, A. Monroy-Hernandez, and P. Irani. Authoring data-driven videos with dataclips. *IEEE transactions on visualization and computer graphics*, 23(1):501–510, 2016. 1, 3
[10] S. Bateman, R. L. Mandryk, C. Gutwin, A. Genest, D. McDine, and C. Brooks. Useful junk? the effects of visual embellishment on comprehension and memorability of charts. In *Proceedings of the SIGCHI conference on human factors in computing systems*, pp. 2573–2582, 2010. 2
[11] A. F. Blackwell, C. Britton, A. Cox, T. R. Green, C. Gurr, G. Kadoda, M. S. Kutar, M. Loomes, C. L. Nehaniv, M. Petre, et al. Cognitive dimensions of notations: Design tools for cognitive technology. In *Cognitive Technology: Instruments of Mind: 4th International Conference, CT 2001 Coventry, UK, August 6–9, 2001 Proceedings*, pp. 325–341. Springer, 2001. 9
[12] L. Blazer. *Animated Storytelling*. Peachpit Press, 2019. 1, 3
[13] M. A. Borkin, A. A. Vo, Z. Bylinskii, P. Isola, S. Sunkavalli, A. Oliva, and H. Pfister. What makes a visualization memorable? *IEEE transactions on visualization and computer graphics*, 19(12):2306–2315, 2013. 2
[14] M. Bostock, V. Ogievetsky, and J. Heer. D^3 data-driven documents. *IEEE transactions on visualization and computer graphics*, 17(12):2301–2309, 2011. 1, 3, 4
[15] M. Brehmer, B. Lee, B. Bach, N. H. Riche, and T. Munzner. Timelines revisited: A design space and considerations for expressive storytelling. *IEEE transactions on visualization and computer graphics*, 23(9):2151–2164, 2016. 1, 3

[16] H. Cheng, J. Wang, Y. Wang, B. Lee, H. Zhang, and D. Zhang. Investigating the role and interplay of narrations and animations in data videos. In *Computer Graphics Forum*, vol. 41, pp. 527–539. Wiley Online Library, 2022. 7

[17] W. Cui, J. Wang, H. Huang, Y. Wang, C.-Y. Lin, H. Zhang, and D. Zhang. A mixed-initiative approach to reusing infographic charts. *IEEE Transactions on Visualization and Computer Graphics*, 28(1):173–183, 2021. 6

[18] T. Ge, B. Lee, and Y. Wang. Cast: Authoring data-driven chart animations. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, pp. 1–15, 2021. 3

[19] T. Ge, Y. Zhao, B. Lee, D. Ren, B. Chen, and Y. Wang. Canis: A high-level language for data-driven chart animations. In *Computer Graphics Forum*, vol. 39, pp. 607–617. Wiley Online Library, 2020. 1, 3, 4, 6

[20] L. Grammel, C. Bennett, M. Tory, and M.-A. D. Storey. A survey of visualization construction user interfaces. In *EuroVis (Short Papers)*, pp. 019–023, 2013. 3

[21] S. Haroz, R. Kosara, and S. L. Franconeri. Isotype visualization: Working memory, performance, and engagement with pictographs. In *Proceedings of the 33rd annual ACM conference on human factors in computing systems*, pp. 1191–1200, 2015. 2

[22] J. Heer and M. Bostock. Declarative language design for interactive visualization. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):1149–1156, 2010. 3

[23] J. Hullman, S. Drucker, N. H. Riche, B. Lee, D. Fisher, and E. Adar. A deeper understanding of sequence in narrative visualization. *IEEE Transactions on visualization and computer graphics*, 19(12):2406–2415, 2013. 1

[24] A. Jahanlou and P. K. Chilana. Katika: An end-to-end system for authoring amateur explainer motion graphics videos. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*, pp. 1–14, 2022. 1, 3

[25] A. Jahanlou, W. Odom, and P. Chilana. Challenges in getting started in motion graphic design: Perspectives from casual and professional motion designers. In *Graphics Interface 2021*, 2020. 1

[26] Y. Kim and J. Heer. Gemini: A grammar and recommender system for animated transitions in statistical graphics. *IEEE Transactions on Visualization and Computer Graphics*, 27(2):485–494, 2020. 3, 5

[27] Y. Kim and J. Heer. Gemini 2: Generating keyframe-oriented animated transitions between statistical graphics. In *2021 IEEE Visualization Conference (VIS)*, pp. 201–205. IEEE, 2021. 3

[28] J. Krasner. *Motion graphic design: applied history and aesthetics*. Taylor & Francis, 2013. 1

[29] Z. Liu, J. Thompson, A. Wilson, M. Dontcheva, J. Delorey, S. Grigg, B. Kerr, and J. Stasko. Data illustrator: Augmenting vector design tools with lazy data binding for expressive visualization authoring. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, pp. 1–13, 2018. 3

[30] M. Lu, C. Wang, J. Lanir, N. Zhao, H. Pfister, D. Cohen-Or, and H. Huang. Exploring visual information flows in infographics. In *Proceedings of the 2020 CHI conference on human factors in computing systems*, pp. 1–12, 2020. 2

[31] A. M. McNutt. No grammar to rule them all: A survey of json-style dsls for visualization. *IEEE Transactions on Visualization and Computer Graphics*, 29(1):160–170, 2023. doi: 10.1109/TVCG.2022.3209460 1, 3

[32] W. Sarinastiti, D. Susanto, and R. Mutammimah. Skill level animation technique on dental care motion graphic for children. In *2016 International Electronics Symposium (IES)*, pp. 389–394. IEEE, 2016. 1

[33] A. Satyanarayan, R. Russell, J. Hoffswell, and J. Heer. Reactive vega: A streaming dataflow architecture for declarative interactive visualization. *IEEE transactions on visualization and computer graphics*, 22(1):659–668, 2015. 5, 9

[34] L. Shen, Y. Zhang, H. Zhang, and Y. Wang. Data player: Automatic generation of data videos with narration-animation interplay. *IEEE Transactions on Visualization and Computer Graphics*, 9, September 2023. To appear in Proc. IEEE VIS'23. 3

[35] Y. Shi, X. Lan, J. Li, Z. Li, and N. Cao. Communicating with motion: A design space for animated visual narratives in data videos. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, pp. 1–13, 2021. 1, 3, 4

[36] J. Thompson, Z. Liu, W. Li, and J. Stasko. Understanding the design space and authoring paradigms for animated data graphics. In *Computer Graphics Forum*, vol. 39, pp. 207–218. Wiley Online Library, 2020. 1, 5

[37] J. R. Thompson, Z. Liu, and J. Stasko. Data animator: Authoring expressive animated data graphics. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, pp. 1–18, 2021. 3

[38] Y. Wang, Y. Gao, R. Huang, W. Cui, H. Zhang, and D. Zhang. Animated presentation of static infographics with infomotion. In *Computer Graphics Forum*, vol. 40, pp. 507–518. Wiley Online Library, 2021. 2, 3, 6

[39] Y. Wang, A. Segal, R. Klatzky, D. F. Keefe, P. Isenberg, J. Hurtienne, E. Hornecker, T. Dwyer, and S. Barrass. An emotional response to the value of visualization. *IEEE computer graphics and applications*, 39(5):8–17, 2019. 2

[40] J. Zong, J. Pollock, D. Wootton, and A. Satyanarayan. Animated vega-lite: Unifying animation with a grammar of interactive graphics. *IEEE Transactions on Visualization and Computer Graphics*, 29(1):149–159, 2022. 3