Encodable: Configurable Grammar for Visualization Components

ABSTRACT

There are so many libraries of visualization components nowadays

with their APIs often different from one another. Could these com

ponents be more similar, both in terms of the APIs and common

functionalities? For someone who is developing a new visualization

component, how should the API look like? This work drew inspira

tion from visualization grammar, decoupled the grammar from its

rendering engine and adapted it into a configurable grammar for in

dividual components called Encodable. Encodable helps component

authors define grammar for their components, and parse encoding

specifications from users into utility functions for the implementa

tion. This paper explains the grammar design and demonstrates how

to build components with it.

Index Terms: Information visualization, systems, toolkits, API

design, reusable visualization, visualization component

1 INTRODUCTION

The GrammarofGraphics (GoG)[37] introduced the idea of a single

language which could express all visualizations, rather than thinking

about visualizations as a catalog of charts (line, bar, pie, etc.) Visual

ization libraries such as ggplot2 [36], and later Vega-Lite [32], grew

from this philosophy. This one-grammar-to-define-them-all works

really well for exploratory data analysis, which expressiveness and entific communities. Users can fluidly transform one visualization

into another by adding or modifying a few expressions.

However, the number of libraries that follow the chart-based ap

proach, is still growing strongly in parallel. This includes everything

from a large library with an extensive suite of chart types [1,3,6,10]

to a tiny library with a single unique visualization. There are a few

reasons why they keep growing: First, when a developer already

have a specific chart in mind, picking that chart from a catalog and

setting a few options is more straightforward than learning how to

express the chart via the grammar. Second, when someone is de

veloping a novel visualization technique, or converting a bespoke

visualization into a reusable component, he/she is likely to focus on

just a single component. Third, performance is often an important

factor for application development. A small library that does a few

things really well can be more preferred than a large library that

offers many unused functionalities.

Each of these chart-based libraries defines its own API for cus

tomizing the charts. Most of the time, their APIs are different from

one another. Switching libraries means learning a new API. For

example, to specify how to get a value for x-position from the data:

Some libraries take a field name string. Some accept a lambda

function xAccessor. Some require each data entry to have a field

named x. They also often do not offer the same amount of common

functionalities. For instance, some libraries support logarithmic

scale, while others do not. In some cases, even components within

the same library have this discrepancy. For component users, are

there ways these components can converge in the future to have

similar APIs and common functionalities? For component authors,

if someone wants to develop a new bespoke component or library, is there a recommended way to define its API? From studying Vega

Lite grammar, an idea came to mind. What if we build components

that have APIs similar to it, but can handle the rendering ourselves?

Instead of having a grammar that can define all graphics tightly

coupled with the rendering engine that transforms that grammar

into actual visualizations, what if we decouple the grammar from

the rendering engine and make it shareable among multiple compo

nents? Each component then can configure the shared grammar to

define its scope, use that subset of grammar as its API and handle

the rendering independently.

Expanded from this idea, this work introduces Encodable, a configurable grammar for encoding a component with data. With this, component authors can: (a) Define and customize encoding grammar for each component which conforms to the shared grammar (b) Validate encoding specification from component users according to the defined grammar (c) Parse encoding specification into useful utility functions for implementing the component.

The rest of the paper is organized as follows: The next section

reviews relevant work, followed by an explanation of goals and

requirements. Then the solution is described in Section 4, accompa

nied by a demonstration and a brief discussion before the conclusion. 2 RELATEDWORK

There are many ways to create a visualization. The programmatic

approaches, mainly on the web, can be grouped as follows:

A) Graphics Manipulation: Processing [28,29] and others [11,

13] let a developer draw or interact with visual elements directly.

They have the maximumlevel of expressiveness and in return require

the most effort to produce the same visualizations.

B) Low-level Composition: D3 [21] learns from the early ap

proaches [20,22,23] and introduces low-level building blocks, such

as selection, scales, formatting, etc. It leverages the common stan

dards such as SVG instead of defining all constructs by itself. vx [19]

bridges D3 and SVG for React [14] framework. Visualizations can

be created from very flexible combinations of these building blocks.

C) Visualization Grammar: Heavily inspired by the Grammar

of Graphics [37], there is no concept of chart type. Developers learn

how to express the visualizations they desire in the given grammar,

i.e. a domain-specific language provided by each library that de

scribes how to transform and encode data into visual marks and their

properties [4,27,31–36]. The most famous one is ggplot2 [35,36]

which dominates the R and data science communities. Vega [31]

let users describe visualizations in JSON, and generate interactive

views using either HTML5 Canvas or SVG. Vega-Lite [32] provides

a higher-level grammar equivalent to ggplot2 level with interactions.

D) High-level Composition: Similar to the convention of MS

Excel [9], this group uses series to abstract a group of graphic ele

ments that encode data. For example, bars in a cartesian coordinate

system form a series. More complex combinations such as candle

stick, bullet or other chart types can also be abstracted as a series.

The data and options are often mixed within the series definition.

ECharts [25] and others [7,12] employ the all-in-one JSON option

to declare a visualization. Many libraries such as Victory [18] and

others [2,15,17] provide similar level of abstraction in React syntax,

such as <XYPlot>, <CandleStickSeries>,or<XAxis>, that can be

composed into the desired visualizations.

E)ChartTemplates: GoogleCharts[6]andothers [1,3,8,10,16]

let developers choose a chart type from its catalog, prepare data in

the specified format and plug them together. Some libraries provide

multiple levels of abstraction. For instance, G2Plot [5] provides

chart templates on top of G2 [4] grammar.

Encodable was designed to complement these approaches. It does

not render the output and therefore cannot create a visualization

by itself alone. Instead, it bridges the gap between the component

authors and users. A component author uses Encodable to define

the component API, uses it again to parse the users’ specification into an Encoder, then choose from the approaches A-D, or even E

under the hood for rendering (Section 4). The resulting component

f

its into the chart templates (E) level.

There are also some GUI approaches for creating visualizations

with relevant concepts. Encodable is similar in spirit to Data-Driven

Guides [24] and the followings [26,30] which let users pick visual

properties from any arbitrary shape and encode them with data.

3 GOALS&REQUIREMENTS

This project aims to provide the following convenience:

Component authors, who create reusable components, should

be able to create a component with encoding grammar that conforms

to this Encodable grammar, with little effort required to make the

component support the grammar.

Component users, who use the reusable components, should

benefit from the consistent encoding grammar across components

and standardized features even though the components are from

different authors. To avoid mistakes when providing an encoding

specification (spec) for a component, users should also receive syn

tax verification that the spec is grammatically correct.

The goals above are broken down into the following requirements:

• R1: Provide a configurable grammar for encoding a compo

nent with data. The component author can customize grammar

Gto be tailored for component C as G(C). G(C) is still a subset

of G and ensures consistency across different components even

though they are implemented by different component authors, e.g.

G(C1),G(C2),...,G(Cn) ⊂ G

• R2: Handle specification parsing for the component author.

Parse the specification {S ∈ G(C)} into something that helps with

the component implementation. This will also reduce the incon

sistencies due to implementation of the parser.

• R3: Provide mechanism to verify specification from the com

ponent users. Learning a new grammar can take time and mis

takes are inevitable. Immediate feedback when coding is very

valuable to reduce mistakes from providing invalid specifications.

• R4: The library should be lightweight. For this utility to be a

dependency of any reusable component, it should not be so large

that no one wants to import.

4 PROPOSEDSOLUTION

Agrammar and parser was written in TypeScript (TS), which is a

strict syntactical superset of JavaScript (JS) that adds static typing

and transcompiles to JS. By using TS, the grammar (R1) can be

defined as type definitions and utilize static type checking to compare

incoming specifications against the type definitions. This will ensure

that the component users have specified the specifications that are

grammatically correct (R3). The overall architecture of Encodable

can be seen in Fig. 1. Encodable components assume the datasets

are in tabular format such as:

[ {"kind":"Cat", "count":9}, {"kind":"Dog", "count":11} ]

The code snippets in this paper are simplified for explana

tion purposes and may omit some details for brevity. Please

see the supplementary materials for more details or repository

(github.com/kristw/encodable) for the full and latest implementation.

4.1 The Grammar

The first principle of Encodable is each visualization has one or more

channels to encode data, such as color, x, y, etc. For example, a

simple word cloud component has size and text channels. If there is a

grammar to describe what size and text can be, one can describe how

to encode this word cloud component with the given data based on

these two channels. Hence, in its simplest form, Encodable grammar

is defined as key-value pairs of channel names and their definitions.

4.1.1 Channel Definition

This work was heavily inspired by Vega-Lite, which includes channel definitions as part of its grammar. Its grammar is also pure JSON and can be serialized into a simple text file. In Vega-Lite,this is how to encode a bar chart that shows number of each animal:

1 const vegaLiteBarSpec = { "mark": "bar",

2 "encoding": {

3 "x":{"field": "kind", "type":"ordinal"},

4 "y": {"field": "count","type": "quantitative"}}};

In the example above, the first channel name is x and its channel definition is {"field": "kind","type": "ordinal"}, telling the rendering engine to encode the kind field for x-position and count field for y-position, or bar height. Encodable adopts a subset of grammar from Vega-Lite for channel de finition (Channel Def).

1 interface ValueDef{

2 value: number | string | boolean | Date | null; }

3 interfaceFieldDef {

4 field: string; format?: string; title?: string; }

5 interface ScaleFieldDef extends FieldDef {

6 type: 'quantitative' | 'ordinal' | 'temporal' | 'nominal'

7 scale?: ScaleDef; /\*See Supp.Materials\*/ }

8 interface PositionFieldDef extends ScaleFieldDef {

9 axis?: AxisDef; /\*See Supp.Materials\*/ }

10 type ChannelDef =

11 ValueDef | FieldDef | ScaleFieldDef | PositionFieldDef;

According to the grammar defined above, a channel definition can be one of the followings:

1. Fixedvalue(ValueDef) – such as making color of text in a word cloud always red.
2. Dynamic value based on a field in the data (FieldDef) such as using the field kind for each word in word cloud.
3. Dynamic value with scale (ScaleFieldDef) – Many channels use scale to map input value into output such as mapping kind into color, count into fontSize. Inside the scale field, the component users can define how they want to customize the scale. The type field in channel definition will help the filler choose the appropriate scale or format when not specified. E.g., a quantitative field uses a linear scale with number formatter by default while a temporalfieldusesatimescalewithtimeformatterbydefault.Thetwoscale

typeshandleticksanddomainroundingdifferently.

(d)Dynamicvaluewithscaleandaxis(PositionFieldDef)

Channelssuchasxorycanoptionallyincludedefinitionforaxes.

1 constcolor:ValueDef = {value:'red' };

2 consttext:FieldDef ={field: 'kind' };

3 constcolor:ScaleFieldDef = {

4 type: 'nominal', field:'kind',

5 scale:{ type: 'ordinal',range: ['pink','blue']} };

6 constfontSize:ScaleFieldDef = {

7 type:'quantitative',field:'count',

8 scale:{ range: [0, 36]}};

9 consty:PositionFieldDef = {

10 type:'quantitative',field:'count',

11 scale:{ nice:true }, axis:{orient:'left' }};

4.1.2 DefineComponent-specificChannels

Atthetimeofthiswriting,Vega-Litehas35channels(x,y,color,

etc.)Evenso,therearestilledgecasesthatarebeyondthesefixedset

ofchannels.Forexample,ifthedeveloperistryingtoencodedata

intofont-family,thereisnosuchchannelinVega-Liteandtherefore

youcannotuseit. Soafixednumberofchannelsdoesnotsound

likeagoodidea. EarlierinSection4.1.1,Encodablegrammaris

definedbroadlyasakey-valueobject (Encoding)withkeybeing

channelnameandvaluebeingchanneldefinition. Thisbasically

allowsunlimitednumberofchannels.

interfaceEncoding{ [channelName:string]:ChannelDef}

However, thisistooambiguousandproblematic. channelName

canbeanystring. Thereisnothingtoenforcecomponentusers

tospecifythecorrectchannelnames,whichbasicallyviolatesR3.

Usersmayspecifychannelcolorwhenthereisnosuchchannelin

thecomponent.Alsoeachchannelmaysupportonlyasubsetofthe

ChannelDeftype. E.g.,atextchanneldoesnotcareaboutaxisor

scaleandshouldonlybeValueDeforFieldDef.

Therefore,thesecondprincipleofEncodableisthecomponent

authorscandefinechannelnamesanddefinitionsspecifictotheir

componentsviaaconfigurationbelow.

1 typeChannelType='X'|'Y'|'Numeric'|'Category'|'Color'|'Text';

2 typeOutput=number|string|boolean |null;

3 interfaceEncodingConfig{

4 [name:string]:[ChannelType,Output,'multiple'?];

5 }

Componentauthorsmustlisttheirchannelnameswiththeirtypes,

expectedoutput type,andwhether itcantakemultiple(arrayof)

definitions(suchasatooltipchannelcanacceptmultiplefieldsto

bedisplayed).Forexample,tocreateawordcloudcomponentthat

canbeencodedbycolorandfontsizeandacceptmultiplefieldsfor

tooltip,thecomponentauthorwillwritethisconfiguration(Fig.1-A)

andderivetheencodinggrammarfromtheconfig(Fig.1-B).

1 import{DeriveEncoding}from'encodable';

2 interfaceWordCloudConfig{

3 color:['Color',string];

4 fontSize:['Numeric',number];

5 text:['Text', string];

6 tooltip: ['Text',string, 'multiple'] }

7 typeWordCloudEncoding=DeriveEncoding<WordCloudConfig>;

InDeriveEncoding(Fig.1-B),eachChannelTypeintheconfigis

mappedtoanappropriatesubsetofchanneldefinitionasfollows:

ChannelType ChannelDefinition

X,Y PositionFieldDef|ValueDef

Numeric, Category,Color ScaleFieldDef|ValueDef

Text FieldDef|ValueDef

XandYchannel typesrepresentx-andy-positions. Numeric

channeltypemeansanumericattribute,e.g.,size,opacity.Category

channel typedefinesacategoricalattribute,e.g.,visibility,shape.

Colorchanneltypedefinesacolorattribute,e.g.,fill,stroke.Text

channeltypedefinesaplaintextattribute,e.g. tooltip, label.The

grammarWordCloudEncodingderivedfromtheWordCloudConfig

isequivalent tothemanually-definedWordCloudEncodingbelow.

However,theextrainformationinconfigthatachannelisaColor

type,notanordinaryCategorywillbeusefulduringparsing,which

themanualonecannotcapture.

1 typeWordCloudEncoding={

2 color:ValueDef |ScaleFieldDef<string>;

3 fontSize:ValueDef |ScaleFieldDef<number>;

4 text:ValueDef |FieldDef<string>;

5 tooltip: (ValueDef|FieldDef<string>)[];/\*array\*/}

4.2 TheEncoder

Encodabletakesencodingconfig(Fig.1-A)fromtheauthorand

encodingspecificationfromtheuser(Fig.1-F),andparsesitinto

anEncoder(Fig.1-K) thatencapsulatesthelogichowtoencode

eachchannel fromdata(R2). Duringparsing,eachchanneldef

initionisparsedseparately. Sincemanyfieldsareoptional, the

filler(Fig.1-I)willexpandtheincomingdefinitionintoacompleted

definitionviasmartdefaultsandinference.Afterthat,eachchan

neldefinitionisparsedintoaChannelEncoder(Fig.1-J),which isautilityclass thatprovidesseveraluseful functions, suchas:

encodeDatum(datum)whichconvertsinputdatumintooutputvalue

forthatchannelandgetValueFromDatum(datum)whichreturnsthe

rawfieldvaluefrominputdatum,orfixedvalue,forthatchannel.

AllChannelEncoderinstancesarenestedunderanEncoderinstance

andreferredtobyencoder.channels[channelName]. Theauthor

thencanusetheEncoderandtheseChannelEncodertohelpwiththe

rendering(Fig.1-L)ofthevisualization.

TheEncodable library, at the timeof thiswriting, is25.2kB

(minified),whichisrelativelysmall(R4). Incomparison,Vega-Lite

### is237.1kB,G2is414.9kBandEchartsis817kB. Tóm tắt bài báo về Encodable

**1. Giới thiệu**

Encodable là một thư viện TypeScript hỗ trợ định nghĩa và xử lý các kênh mã hóa dữ liệu trong trực quan hóa. Được lấy cảm hứng từ Vega-Lite, Encodable giúp tiêu chuẩn hóa cách biểu diễn dữ liệu mà không phụ thuộc vào nền tảng vẽ (React, D3, SVG, v.v.).

**2. Giải pháp đề xuất**

Encodable sử dụng cú pháp TypeScript để kiểm tra tính hợp lệ của thông số đầu vào, đảm bảo rằng người dùng định nghĩa các kênh mã hóa chính xác. Dữ liệu được giả định ở dạng bảng, ví dụ:

[{"kind":"Cat", "count":9}, {"kind":"Dog", "count":11}]

**3. Ngữ pháp của Encodable**

Encodable định nghĩa một biểu đồ thông qua các **kênh mã hóa** như x, y, color, text, v.v. Mỗi kênh có thể có một trong các kiểu giá trị:

* **Giá trị cố định** (ValueDef) – ví dụ: màu chữ luôn đỏ.
* **Giá trị động dựa trên dữ liệu** (FieldDef) – ví dụ: hiển thị loại động vật.
* **Giá trị động với thang đo** (ScaleFieldDef) – ánh xạ dữ liệu vào không gian hiển thị.
* **Giá trị động có cả thang đo và trục** (PositionFieldDef) – định nghĩa trục tọa độ.

Ví dụ, cách định nghĩa thanh màu theo trường dữ liệu:

const color: ScaleFieldDef = {

type: 'nominal', field: 'kind',

scale: { type: 'ordinal', range: ['pink', 'blue'] }

};

**4. Định nghĩa kênh mã hóa theo thành phần cụ thể**

Thay vì giới hạn số kênh như Vega-Lite (có 35 kênh cố định), Encodable cho phép mỗi thành phần định nghĩa kênh riêng bằng TypeScript:

interface WordCloudConfig {

color: ['Color', string];

fontSize: ['Numeric', number];

text: ['Text', string];

tooltip: ['Text', string, 'multiple'];

}

Điều này giúp tùy biến linh hoạt, đồng thời đảm bảo người dùng không nhập sai tên kênh.

**5. Bộ mã hóa (Encoder)**

Encodable chuyển đổi thông số từ người dùng thành bộ mã hóa (Encoder), giúp đơn giản hóa quá trình hiển thị dữ liệu:

const encoder = createEncoder<WordCloudConfig>(encoding);

encoder.setDomainFromDataset(data);

Ví dụ, khi hiển thị đám mây từ (WordCloud), bộ mã hóa tự động tính toán màu sắc, kích thước chữ dựa trên dữ liệu đầu vào.

**6. Trình diễn và ứng dụng thực tế**

Encodable được sử dụng để xây dựng các biểu đồ như bản đồ Trung Quốc, biểu đồ cốc cà phê, và nhiều biểu đồ trong Apache Superset. Thư viện này giúp dễ dàng thêm/bớt kênh mã hóa mà không làm thay đổi logic hiển thị.  
Ví dụ:

jsx

Sao chépChỉnh sửa

<ChinaMap data={data} encoding={{

location: { field: 'province' },

fill: { field: 'numStudents', type: 'quantitative' }

}} />

Encodable hiện có trên npm với **3,000 lượt tải mỗi tuần**.

**7. Kết luận và hướng phát triển**

Encodable giúp tiêu chuẩn hóa API của các thành phần trực quan hóa, mở rộng khả năng tùy chỉnh mà vẫn giữ được sự nhẹ nhàng (chỉ 25.2kB, nhỏ hơn nhiều so với Vega-Lite, G2, ECharts). Trong tương lai, thư viện sẽ được mở rộng để hỗ trợ **legend, trục tọa độ, và nhiều tính năng mới**.

**8. Lời cảm ơn**

Cảm ơn nhóm Airbnb Data Experience và các nhà nghiên cứu đã đóng góp ý kiến cho dự án.

4o