Cubbles

A platform to develop and share client-side components for webapplications.

Hans-Dieter Böhlau
incowia GmbH, Albert-Einstein-Str. 3, 98693 Ilmenau, Germany

Keywords: software development, webcomponents, client-side webapps

Abstract

Cubbles is a platform to develop and provide client-side components for webapplications. The platform is currently under development with financial support of the German Federal Ministry for Education and Research (BMBF) within the project WEBBLE TAG[4]. Subject of this paper is to give a summary of the vision and the current state of development. At the beginning an outline of motivation and vision behind Cubbles will be given. The next part is about the scope of the platform and it's position in relation to other approaches and frameworks. Within a third chapter some information about the basic architecture and the main-features will be provided and finally the reader willl find an overview of current limitations and the roadmap.

Motivation and Vision

Since the arise of HTML5 webbrowsers evolve towards a powerful runtime environment for complex applications. We are able to move a large part of server-side logic onto the client. This powers new usage scenarios, e.g. offline usage, client-side data processing etc.
And very interesting from the developers point-of-view, it simplifies the development of much more sophisticated webapplications without implementing backend-located functionality. Especially if applications are provided for a larger audience, the operation of backend infrastructures tends to be complex and needs another skillset.

But nevertheless, building complex client-side webapplications is a new thing - compared to the number of years we build "traditional" desktop-applications or server-side systems. The basics did not change with HTML5 - the browser does processing the DOM to create the user-interface and deals with javascript and css for logic and styling stuff. If you want to structure your application, encapsulate resposibilities to dedicated parts of the application and integrate those parts with each other, neither webbrowser nor HTML5 do have any supporting concepts implemented. Maintaining a more complex application or integrate already existing functionality is difficult. Application-frameworks like ANGULARJS[12] and EMBERJS[13] provide support to build applications using of a component-based approach; but those components are re-usable within the particular frameworks only.

Inspired by the idea of webbles as "interactive digital objects" (KUWAHARA,TANAKA[1]) and with the familiar principles of component-based software engineering (WIKI-SBSWE[3]) in mind, we started to develop the idea of a more open platform. It should allow people to create and share components for re-use within almost any webapplication.

Creating components means to easily create them from scratch (code based) or to compose them from existing components. Shared components are available for other developers to create more complex components or to directly use them within their webapplications. Existing platforms like Github already proved, the idea of shared and evolutionary development increases the efficiency to create software-based solutions for many kinds of problems.

Our vision is about components and a platform with related tools and services:

Components

- can play with each other independently of the framework they have been implemented with,
- can be used to compose more complex (compound-) components,
- can easily replaced by newer versions of a component or other components (with a matching interface),
- are simple to integrate into new and existing webapplications.

The platform

- provides tools supporting the development of components,
- allows to share and find shared components,
- provides an implementation of a component-interaction-model,
- serves components for direct integration into webapplications,
- is easy to setup in public and private environments.

Scope of Cubbles

What it is not aimed to be:

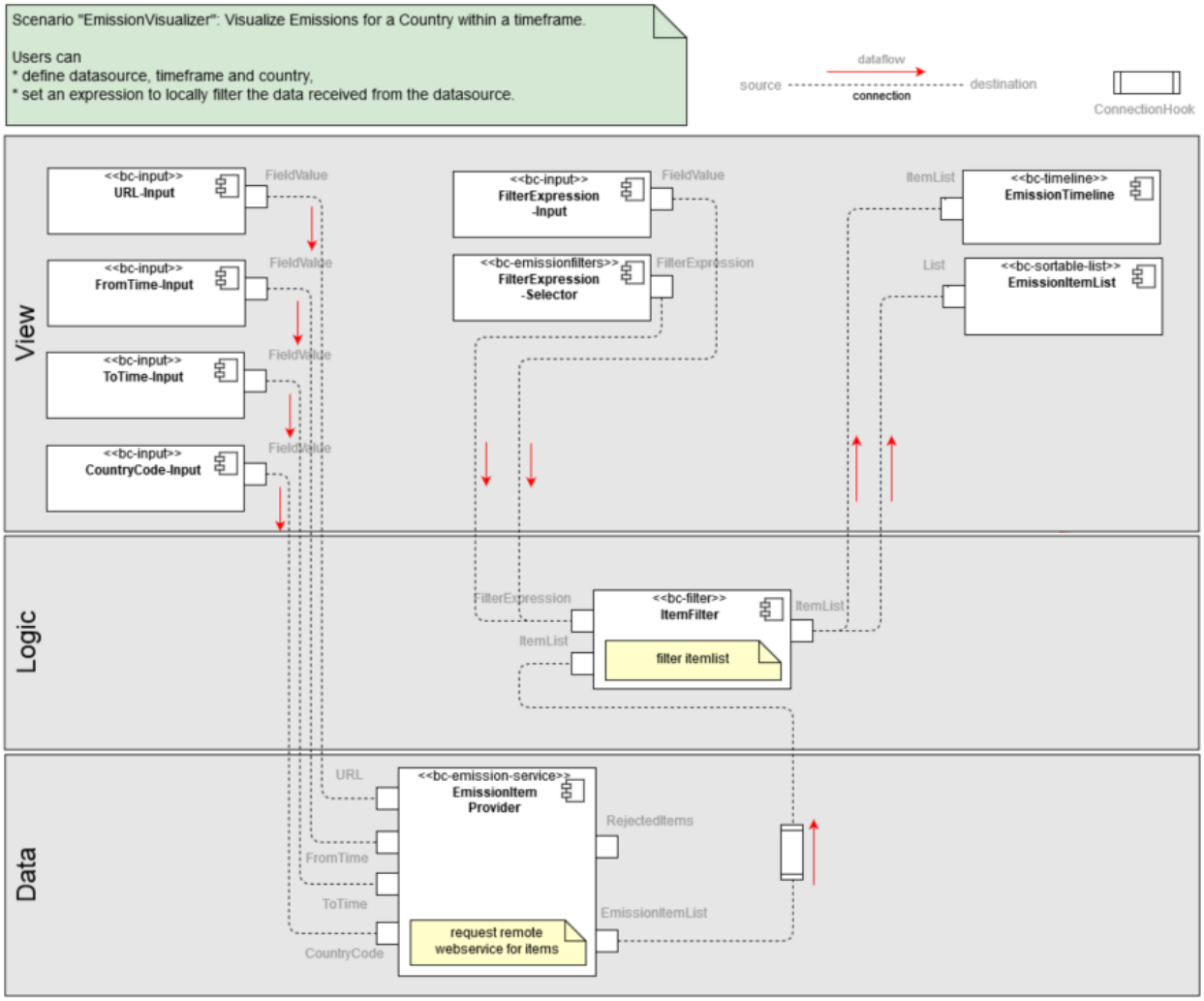
- It's not an application framework. Look at AngularJS etc. or established server-side frameworks.
- It's not a webcomponent framework. Instead Cubbles is designed to adapt different component frameworks and provides an integration-layer enabling interactions between components at runtime. Currently we provide an adapter for Polymer-1.2.3.
- It's not a datastore. Surely your components deal with data. Use data-services like Firebase or Cloudant and create or (re-) use components to connect to those service(s).

Instead, Cubbles does focus the following aspects:

1. Component Interaction: Create components by composing one or more components from a number of existing solutions (components).
2. Component Integration: Integreate 1..n components into a webapplication - independently of the technology used to create the document-object-model (DOM[5])

While for the development of elementary-components we recommend the use of dedicated frameworks and we prefer/commend a model-based approach to declare the interactions of different component-instances with each other. You can choose your prefered way to design your instance-interaction-model. If you like, start with pen and paper to identify components and sketch out the data-flows (connections) between them. Finally the compound-declaration needs to be available within a json file that will be processed on client-side by the so called "Component Interaction Framework" (CIF).

To get an impression, what it means to design an instance-interaction-model, look at the following picture:



You see 10 component-instances, based on 6 components. It's not obvious, if these components are elementaries or compounds - it doesn't matter. You see components with 0..n input-slots on the left side and 0..n output-slots on the right. Connections are defined between slots with a dataflow from output-slot to input-slot. The change of an output-slot-value triggers the transmission of the new value to all connected components (following the order of the connection within the compound declaration). As an input-slot-value-change may trigger an output-slot-value-change, you can create complex sequences that can change the state of the whole system. At composition-time you look at components as blackboxes. You can connect them and you define so-called "connection-hooks". These are javascript functions you can implement to validate or transform the transferred value, before setting the new value on the connected input-slot.

The layered-architecture within the picture above is optional and without representation inside of the compounds declaration. But it shows an implication of the component based approach - you can use common architectural patterns to find a proper design for your system.

As a compound-declaration can be a complex thing, part of the WEBBLE TAG[4] project is the development of a browser-based development environment (BDE) for compound-components. Providing detailed information about the BDE ist not scope of this paper.

Cubbles is not and does not contain an application framework. You are free to create your application with "plain" HTML5 or with support of an application framework of your choice. Cubbles just provides ways to instantiate components within you DOM. The most simple approach, just write the components corresponding html-tag and provide the id of the component. All the rest is done automatically by the so-called "Client Runtime Container" (CRC), which needs to be included via script-tag into your page.

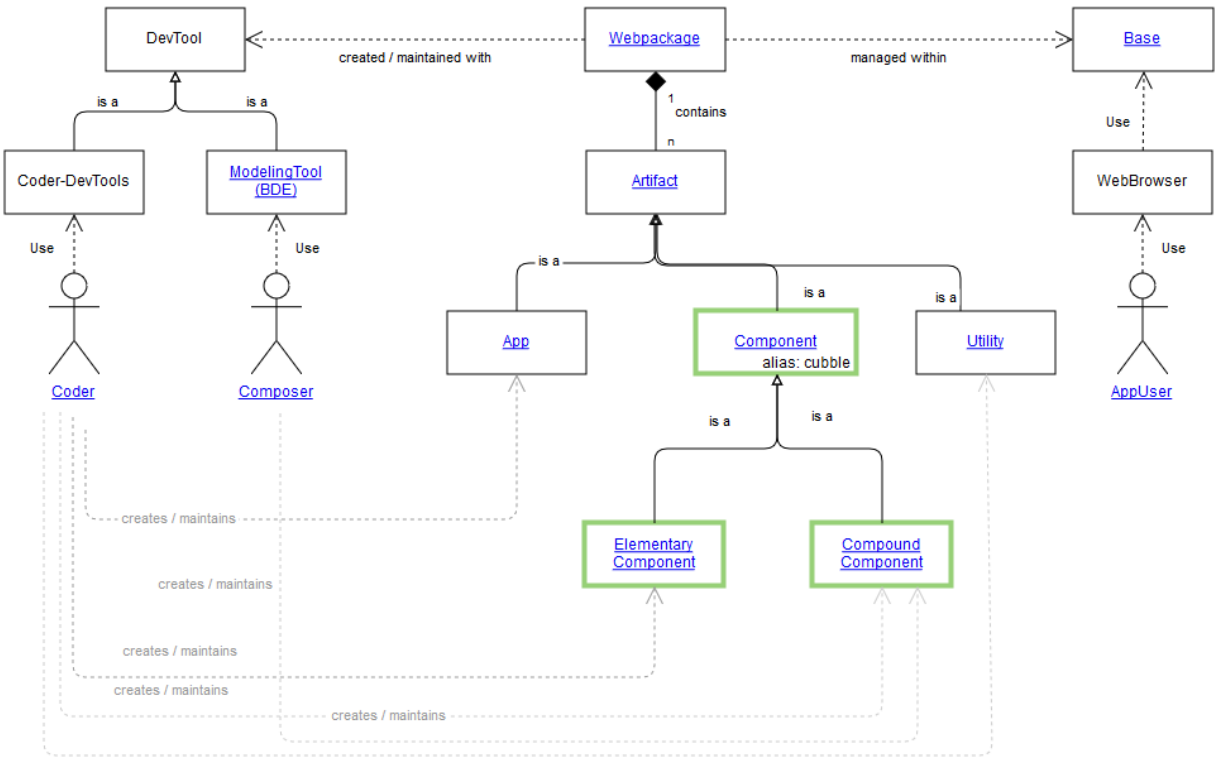
So finally you can describe the approach basically as "Create components and use them on any webpage."

Basic Architecture and Features

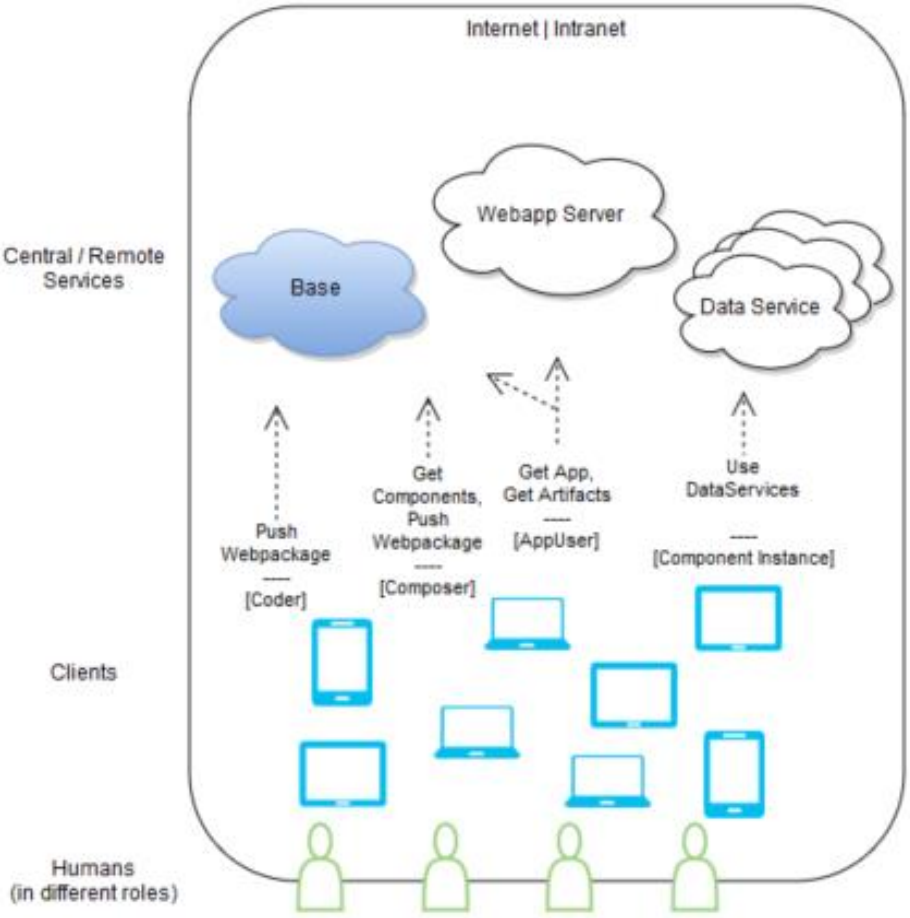
To understand the architecture of Cubbles, let us clarify some concepts in the context of the already mentioned term "component":

1. The first important term to understand the graphics below, is what we call a "Webpackage": See it as a container for webresources, e.g. the files containing the code and description of a component.
2. Another important term is "Base": The Cubbles Base is the server-side located backend, responsible to store webpackages persistently and to deliver the containing webresources, if requested by any client.

Basic concepts, important to understand Cubbles.



A context view. The Base as a central service for webpackage-management and webresource-delivery.



The Client-Side:

We differentiate between 3 user roles - Coder, Composer, AppUser. Coder and Composer create and maintain webpackages, AppUser do use components as parts of webapplications.

- For Coders we provide a set of development tools on top of NODEJS[9]. Use the IDE or editor of your choice and start implementing your webpackages. If you want your first webpackage-version to be available for others, run the provided upload task.
- For Composers - as already mentioned - we are currently working on a browser-based application to support compostion of (compound-) components. The application itself is a client-side webapplication and is provided as a webpackage within a Base -store.
- For AppUsers, the use of the Cubbles platform is something that works under the hood. Except you use the Base to provide a full-fledged client-side webapplication, in that case AppUsers request the Base directly.

Technology basics:

- Composers and AppUsers solely use the webbrowser as the runtime environment. No other installation is needed.
- To integrate Cubbles into a webapplication a so called runtime-extension (RTE) has been developed on top of standard javascript.
  The most important utilities are:
  - Client-Runtime-Container (CRC) ... responsible for transitive dependency resolution and triggering the download of resolved references.
  - Component-Integration-Framework (CIF) ... responsible for creating compound-components, connecting components and managing the dataflow.
  - CubxPolymer ... responsible to adapt POLYMER[10] based components for use with the CIF.

The Server-Side (Base):

The server-side of the platform is called "Base" - responsible to manage webpackages and deliver requested resources from any webpackages to the clients. Webpackages are managed within 1..n "stores". A store is an independant shared-space for webpackages which potentially refer to each other. Each webpackage (-version) within a store and each resource within a webpackage is available under a unique url. As those resources are static and different webpackages can refer to each other, we can simply use the standard caching approaches to minimize download volume in favour of the startup-performance on client-side.

Technology basics:

- On operations-layer, we use DOCKER[6] to abstract from the hosts operation-system and simplify setup and operations. The Base is de-composed into a number of services, each mapped to it's docker-image. This allows to easily scale and/or distribute per service.
- The so-called gateway is based on NGINX[7] and provides the api-layer for webclients. Any client - Base communication is done through the gateway.
- Data persistence is managed by COUCHDB[7]. CouchDB allows us to manage each webpackage (-version) as one document - including all the resources as attachments. Doing so, we can easily replicate webpackages between stores of the same or a distributed Base instance.
- Some other services (e.g. for user-authentication) are implemented on top of NODEJS[9].

Selected Features:

- Base hosting options: If you want the Base to be hosted within your local network, just do so. Replicating webpackages to any store of a remote Base is easy.
- Base supports 1..n stores: As the Cubbles Base supports the creation of 1..n stores, it's possible to create a dedicated space for different kinds of use-cases. Create a sandbox-store for your organisation/team, a store for special kinds of components etc. It's already possible to define who is allowed to upload webpackages on store-level. In future also read-access can be restricted.
- Webpackage versioning: Webpackages are versioned by default (e.g.: my-webpackage@1.0.0). The plattform also allows webpackages to have a snapshot-version (e.g. my-webpackage@1.1.0-SNAPSHOT). Snapshot-versions can be overwritten - a nice feature during development.
- Artifact dependency management and -resolution: Each artifact can have dependencies defined to artifacts of the same and/or of other webpackages. If you refer to an artifact (-version), it's dependencies will be resolved automatically at runtime. This allows to easily re-use existing components and utilities.

- Shared development: The component-based approach in conjunction with the Component Integration Framework allows to easily share development of multiple components with your team-members. As soon as components are identified and their slots and connections are defined, each component can be developed independently. If all your team members upload their work into the same store, you can continuously see the compound-component or app improved better step-by-step.

Current Limitations and Roadmap

There are a lot of things to do. If you'd like to support us, let's get in touch!

Current limitations:

1. Browser support: Currently components do not run in all browsers. The reason for that is, that the WEBCOMPONENTS[2] standard is not implemented in all browsers. Polyfills are provided for many of them, but e.g. for IE and Edge not yet. The Polymer-Team is working on that - e.g. with Microsoft.
2. Only one component per document: Currently we allow to integrate max. 1 component per html-document. This could be an elementary-component or a compound of any complexity.
3. Dependency conflict management: As we want component-developers to refer to exisiting artifacts, this might lead to conflicts if two artifacts (a) and (b) refer to the same artifact (c) ... but to different version (c@1.0) (c@1.1). The winning version finally depends on the implementation of artifact (c). We already have implemented a mechanism to resolve transitive dependencies. We need to extend this by implementing conflict recognition and resolution strategies.
4. User self registration: Currently we do not allow people to register themselves. The reason is simple, we do currently not have the manpower to support a high number of users with unknown skills. But anonymous users can do all, except uploading webpackages into a Base. But for sure, if you contact us - we will create an account and a dedicated store for you and/or your team members.

Of course we will work on all the limitations mentioned above. But it's worth to mention the big points on our roadmap:

1. Composite Case: Currently you compose a compound-component to declare dataflows between components. We want to make that possible on html-tag level to let people define dataflows between any number of integrated components.
2. Component Search: As we now have the component model established, people can create them. If there will be an increasing number of them, it's getting more difficult to find existing appropriate components for (re)use in webapplications or for creating a compound. So we want to provide a powerful search, to support re-use of components.
3. User Management: More sophisticated user-management features are a must-have.
4. React Support: We would like to allow people to implement components based on REACT[8] and to let them interact with Cubbles components based on POLYMER[10].

Acknowledgement

Part of this work has been supported by the German Federal Ministry for Education and Research (BMBF) within the joint project "WEBBLE TAG"[4].

FAQs

1. Cubbles components: Are these the new Webbles?
   No. We do not use the term "Webble", as the concept of Cubbles components is inspired by many different approaches. But yes, webbles are an important source of inspiration and you can build webble-like applications using Cubbles.

<Will be continued, if the paper is accepted.>

References

1. KUWAHARA, Micke Nicander; TANAKA, Yuzuru: Webbles: Programmable and Customizable Meme Media Objects in a Knowledge Federation Framework Environment on the Web. URL: http://ceur-ws.org/Vol-822/MK.pdf
2. WEBCOMPONENTS: WebComponents.org. URL http://webcomponents.org
3. WIKI-CBSWE: Wikipedia: Component-based software engineering. URL: https://en.wikipedia.org/wiki/Component-based_software_engineering
4. WEBBLE TAG: WebbleTAG - Webble Technology Application Group - Ilmenau. URL: http://www.unternehmen-region.de/de/8691.php
5. DOM: Document Object Model. URL: https://de.wikipedia.org/wiki/Document_Object_Model
6. DOCKER: An open platform for distributed applications for developers and sysadmins. URL: https://www.docker.com/
7. NGINX: nginx [engine x] is an HTTP and reverse proxy server. URL: http://nginx.org/
8. COUCHDB: Apache CouchDB™ is a database that uses JSON for documents, JavaScript for MapReduce indexes and regular HTTP for its API. URL: http://couchdb.apache.org/
9. NODEJS: Node.js® is a JavaScript runtime built on Chrome's V8 JavaScript engine. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient. URL: https://nodejs.org
10. REACT: A JavaScript library for building user interfaces. URL: http://facebook.github.io/react/
11. POLYMER: The Polymer library is designed to make it easier and faster for developers to create great, reusable components for the modern web. URL: https://www.polymer-project.org/1.0/
12. ANGULARJS: HTML enhanced for web apps! URL: https://angularjs.org/
13. EMBERJS: A framework for creating abitious web applications. URL: http://emberjs.com/