RWTH Aachen University
Chair for Data Management and Data Exploration
Prof. Dr. T. Seidl

Proseminar

Dart

A brief introduction

David Stutz

April 2012

Mentoring: Prof. Dr. T. Seidl
Dipl. Ing. Marwan Hassani

I declare that I have only used the sources which are listed in the bibliography and that the submitted work has been done without outside help.

Aachen June 29, 2012

Contents

Listings

List of Figures

1 Abstract

Dart is a brand new language for client and server side web development from Google. The language provides class-based object-orientation and allows developing modular and structured applications.

For client side development Google learned from Javascript and provides an own high featured library for DOM (Document Object Model[1]) manipulation and event handling. This paper will introduce client side development with Dart by developing a small sample application.

Allowing server side development, too, Dart allows to create homogeneous systems covering both client and server.

As conclusion Dart shows potential for challenging modern web development. The one question remaining for client-side development using Dart is the following: **Will the browsers implement native support for Dart** – will Dart challenge a growing community for Javascript?

[1] `http://en.wikipedia.org/wiki/Document_Object_Model`

2 Introduction

Dart (first called Dash) is a class-based, object-oriented programming language for the web developed by Google and published under the open source "New BSD License"[2]. [5]

This paper will give a brief introduction to client-side development with Dart by developing a small contact manager written in Dart. Some features and language concepts of Dart will be used and explained.

[2] `http://www.opensource.org/licenses/bsd-license.php`

3 Motivation

With rising complexity of web applications there is a need for writing more structured and modular code.

Can Dart help writing great web applications?

There are developers seeing the need for Dart especially as better alternative to Javascript. Google itself states in a leaked memo:

> "Javascript has fundamental flaws that cannot be fixed merely by evolving the language." [4]

But there is even more than client-side development:

> "Developers have not been able to create homogeneous systems that encompass both client and server, except for a few cases such as Node.js and Google Web Toolkit (GWT)." [1]

Let's have a look at the key features Google points out: [1]

Classes Class-based object-orientation is popular and known to most developers. [1]

Optional typing The programmer can decide on the need for typing, allowing rapid prototyping – without typing – and providing "type-checking tools [...] for debugging" – with typing. [1]

Libraries Software can rely on "independently developed pieces of code". [1]

Tools "Dart will include a rich set of execution environments, libraries, and development tools built to support the language". [1]

4 An example: contact manager

The contact manager will provide the following features:

- List all available contacts.
- Add new contacts.

4.1 Tools

Google provides some useful tools for developing with Dart.

First there is the Dart editor based on Eclipse[3] supporting syntax highlighting and autocompletion for Dart as shown in Figure 1.

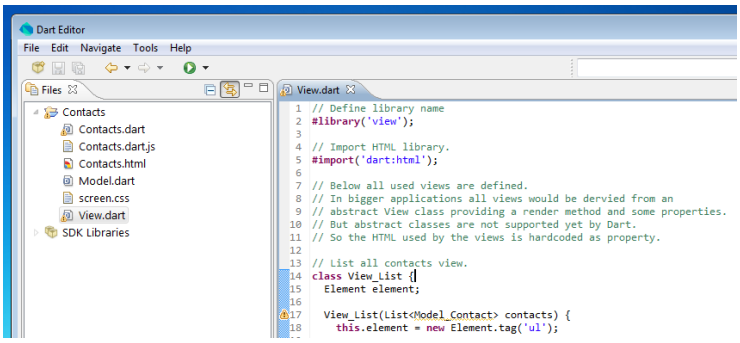

Figure 1: Dart editor with syntax highlighting

For running Dart applications the code is translated to Javascript, so the application can be started in your default browser.

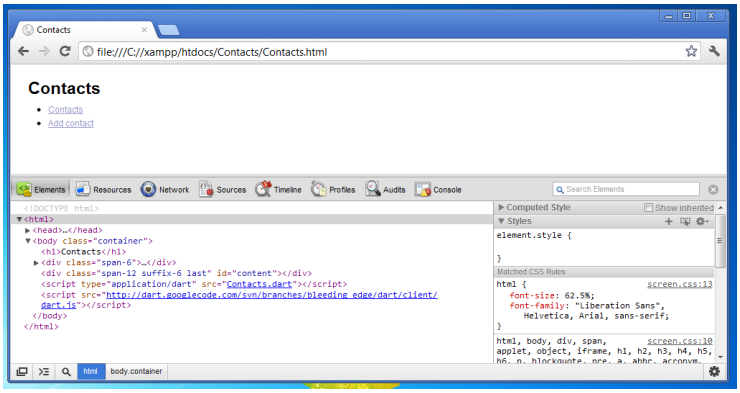

Figure 2: Dartium

[3] `http://www.eclipse.org/`

Dartium[4] is a chromium-based browser – see Figure 2 – and provides the Dart VM. Using Dartium Dart applications can be run without compiling to Javascript.

4.2 Start

Listing 1 shows the code automatically gernerated by the Dart editor after creating a new project:

Listing 1: Basic Dart program

```
#import('dart:html');

class Contacts {

  Contacts() {
  }

  void run() {
    write("Hello World!");
  }

  void write(String message) {
    // the HTML library defines a global "document" variable
    document.query('#status').innerHTML = message;
  }
}

void main() {
  new Contacts().run();
}
```

- The Dart HTML library is imported using the import keyword.
- write() will append a given String to the DOM (Document Object Model[5]) element with id "status". The method is optional typed by void.
- document.query('...') searches a DOM element.
- void main() is the entry point of the script.

Figure 3 shows the output after running the application the first time.

4.3 Modularity

Although it is not necessarily needed, the application will be splitted up in multiple libraries according to the MVC (Model – View – Controller[6]) design pattern.

[4] http://www.dartlang.org/dartium/index.html
[5] http://en.wikipedia.org/wiki/Document_Object_Model
[6] http://de.wikipedia.org/wiki/Model_View_Controller

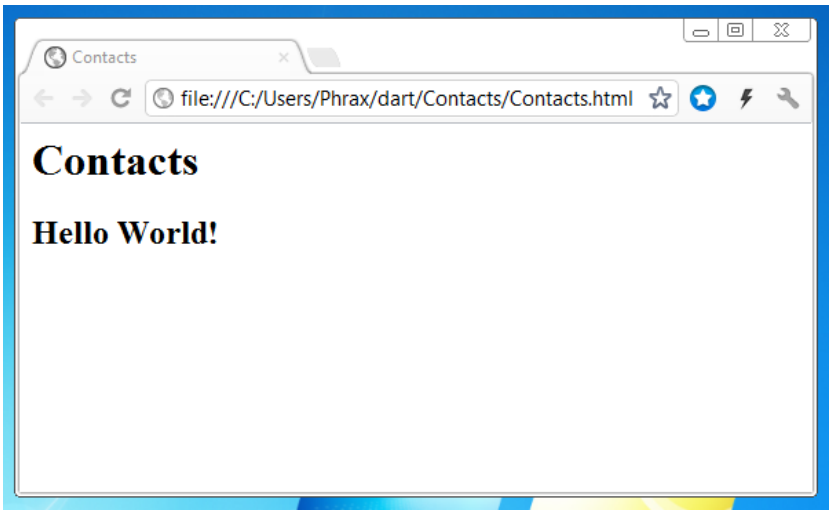

Figure 3: First run of the application

For adding and querying all contacts the class Model will provide add(Model_Contact contact) and find_all (). A single contact is represented by Model_Contact.

Listing 2: Model

```
1  // Define library name.
2  #library('model');
3
4  // On bigger applications an ORM system or similar would be used.
5  // For simplicity the Model class does only need to manage contacts.
6  class Model {
7
8    static List<Model_Contact> contacts;
9
10   // Add a new contact to the list.
11   static void add(Model_Contact contact) {
12
13   }
14
15   // Returns a list of all contacts.
16   // Currently the contacts are hard coded for simplicity.
17   // They could also be loaded from a server via AJAX.
18   static List<Model_Contact> find_all() {
19
20   }
21 }
22
23 // Model for single contact.
24 class Model_Contact {}
```

- #library('model') will define a new library named "model".
- All contacts will be saved in static List<Model_Contact> contacts.

The class "Contacts" will be the controller. The actions for the navigation will be implemented as event handlers in the run() method as seen in Listing 3.

Listing 3: Controller

```
// Main application class with initialization and all needed actions.
class Contacts {

  Contacts() {
  }

  // Will do all the initialization
  // and set up all needed event handlers.
  void run() {
    // Event handler for adding a new contact.
    var add_handler = (Event event) {

    };

    // Event handler for listing all contacts.
    var list_handler = (Event event) {

    };
  }
}
```

- Functions can be assigned to variables – both event handlers are assigned to the appropriate variables add_handler and list_handler .

The views are implemented as independant classes. They could also be derived from an abstract class "View" but for now Dart does not support abstract classes:

> "Dart will support abstract classes and abstract methods. As of 2012-04-04, abstract is not yet implemented." [2]

Listing 4 shows the declarations of the views we will use throughout the application.

Listing 4: Views

```
// Define library name
#library('view');

// Import HTML library.
#import('dart:html');
```

```
6
7  // The view used to list all contacts.
8  class View_List {
9    // Constructor.
10   Element element;
11
12   View_List () {
13     // Create HTML...
14   }
15 }
16
17 // Add a new contact view.
18 class View_Add {
19   Element element;
20
21   // Constructor.
22   View_Add () {
23     // Create HTML...
24   }
25 }
```

- #library('view') defines the library for the views.
- The constructors of the views will build up the HTML.

4.4 Contact model

A contact will have the following attributes with the given types:

- ID – num.
- First name – String.
- Last name – String.
- Phone number – String.
- Email address – String.

Listing 5 shows the typed implementation of these attributes.

Listing 5: Class attributes

```
1  // Model for single contact.
2  class Model_Contact {
3
4    // Attributes are automatically declared private
5    // because of the prefixed underscore.
6    num _id;
7    String _first_name;
8    String _last_name;
```

```
  String _phone;
  String _email;
}
```

- All attributes are declared private by prefixing the underscore – making public and private keywords redundant.

Dart is an optional typed language and supports multiple built-in types[7]. To declare untyped variables the var keyword can be used.

Because all attribtues in the model are declared as private we have to implement getters and setters to get access.

Listing 6: Getters and Setters

```
// Getters and setters for all attributes.
// Note the shorthand syntax used.
String get first_name() => this._first_name;
set first_name(String first_name) => this._first_name = first_name;

String get last_name() => this._last_name;
set last_name(String last_name) => this._last_name = last_name;

String get phone() => this._phone;
set phone(String phone) => this._phone = phone;

String get email() => this._email;
set email(String email) => this._email = email;
```

- Using the get keyword getters can be declared.
- The set keyword is the equivalent for declaring setters.
- In Dart all kind of methods can be declared using a shorthand syntax using =>.

The attributes can now be accessed as if they were public attributes – see Listing 7.

Listing 7: Using getters and setters

```
// The attributes can be handled as if they were public.
contact.first_name = "David";
contact.last_name = "Stutz";
```

Another feature not necessarily known from other class-based languages: named constructors – Listing 8.

[7] `http://www.dartlang.org/language-tour/`

Listing 8: Named constructors

```
// Creates a contact from an id and a map of values,
// A good example for named constructors.
Model_Contact.values(num id, Map<String, String> values) {
    this._id = id;
    this._first_name = values['first_name'];
    this._last_name = values['last_name'];
    this._phone = values['phone'];
    this._email = values['email'];
}
```

- Model_Contact.values(num id, Map<String, String> values) declares the constructor named "values".

Maps and lists are provided by Dart's collection framework – similar to the Java Framework. Implementations of the collection framework can be found in the API reference[8].

4.5 Views

For DOM element creation – Listing 9 – there are named constructors provided. And manipulating DOM elements is simple:

Listing 9: DOM manipulation

```
// Creation by HTML:
Element element = new Element.html("<div>My_element</div>");
// And by tag:
Element element = new Element.tag('div').innerHTML = "My_element";
// Setting attributes:
element.attributes["readonly"] = true;
// Or classes:
element.classes.add('sidebar');
// Add further child elements:
element.nodes.add(new Element.html('<strong>Important!</strong>'));
// Remove all child elements:
element.nodes.clean();
```

- Element creation can be done using the shown named constructors – by tag or by HTML.
- Through attributes and classes the element's attributes and classes are accessible.
- All child nodes of an element are – similar to attributes and classes – kept in a map allowing to remove nodes or add new ones.

[8] http://api.dartlang.org/dart_core.html

4.5.1 Adding a contact

Let's create a form to add a new contact. Of course we could hard code the HTML, but Listing 10 shows another – maybe more circumstantial – way of creating the form.

Listing 10: Form creation

```
// Add a new contact view.
class View_Add {

  Element element;

  View_Add() {
    this.element = new Element.tag('table');

    Map<String, String> rows = {
      'first_name': 'First name',
      'last_name': 'Last name',
      'phone': 'Phone',
      'email': 'E-Mail',
    };

    rows.forEach(function(key, value) {
      // Create the tr node.
      Element row = new Element.tag('tr');

      // Create label and input cells.
      Element label = new Element.html('<td><label
          for="$key">$value</label></td>');
      Element input = new Element.html('<td><input type="text"
          name="$key" id="$key" /></td>');

      // Append label and input cells to row.
      row.nodes.add(label);
      row.nodes.add(input);

      // Append row to table.
      this.element.nodes.add(row);
    });

    // Create last row.
    // The last row contains only one cell with colspan two and a submit button.
    Element row = new Element.tag('tr');
    Element submit = new Element.tag('td');
    submit.attributes['colspan'] = '2';
    submit.innerHTML = '<button type="submit"
        id="submit">Submit</button>';
    row.nodes.add(submit);

    // Append row to table.
```

```
41      this.element.nodes.add(row);
42    }
43  }
```

- First a public attribute element is declared. The view's HTML will later be accessed through this attribute.
- Instead of hard coding each row it is built up while iterating over the declared map. It contains the attribute name as key and the label shown for the input field as value.
- For each row two cells are added: one for the label, one for the input field.
- At the end a last row containing the submit button is appended to the table.

Figure 4 shows the result.

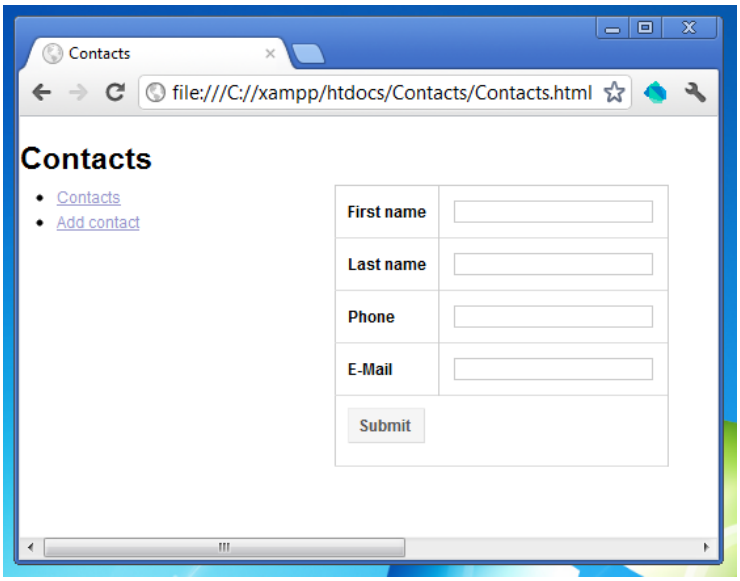

Figure 4: Form for adding a new contact

4.5.2 Listing contacts

The remaining view will list all contacts using an unordered list (<ul>) – see Listing 11.

Listing 11: Listing all contacts

```
1  // List all contacts view.
```

```
class View_List {
  Element element;

  View_List(List<Model_Contact> contacts) {
    this.element = new Element.tag('ul');

    for (Model_Contact contact in contacts)
    {
      // See the usage of getters and setters:
      this.element.nodes.add(new Element.html('<li>${contact.last_name}, 
          ${contact.first_name} - ${contact.phone} - ${contact.email}</li>'));
    }
  }
}
```

- The constructor expects a list of contact models to build the view.
- Dart supports a foreach loop to iterate over all elements of the list.
- For each contact an <li> element is added containing last and first name of the contact.

4.6 Controller

To get access to the views and the model we have to import their libraries:

Listing 12: Importing views and model

```
// Import views.
#import('View.dart');

// Import model.
#import('Model.dart');
```

- The #import declarative expects the relative path to the libraries.

The run() method of the contacts controller will make initialization and set up all event handlers of the navigation which will be hard coded.

Listing 13: Controller initialization – the run() method

```
// Will do all the initialization
// and set up all needed event handlers.
void run() {

  // A good example for event handling.
  // Will bind the navigation action to the appropriate event handler.
  // For querying CSS selectors are used.
  document.query('#list').on.click.add(list_handler);
  document.query('#add').on.click.add(add_handler);
}
```

- Adding and removing event handlers is done by using the add() or remove() method on the chosen event – here on.click.

4.6.1 Adding a contact

So let's set up the event handler for adding a new contact – Listing 14.

Listing 14: add_handler event handler

```
// Event handler for adding a new contact.
var add_handler = (Event event) {

  // Prevent default action of link similar to JQuery.
  event.preventDefault();

  View_Add view = new View_Add();

  // Clean content area before appending the list:
  document.query('#content').nodes.clear(); // Equivalent to JQuery's empty().
  document.query('#content').nodes.add(view.element); // Equivalent to JQuery's
      append().

  // Bind submit event (no real on.submit but on.click).
  document.query('#submit').on.click.add((event) {

    Model.add(new Model_Contact.values({
      'first_name': document.query('input#first_name').value,
      'last_name': document.query('input#last_name').value,
      'phone': document.query('input#phone').value,
      'email': document.query('input#email').value,
    }));
  });
};
```

- Similar to JQuery event.preventDefault() prevents the default action of being executed – here changing the location due to the link clicked.
- Then the view is initialized. The content is cleared using nodes.clear() and the view element is appended.
- For adding the new contact an event listener on.click is set, so Dart can handle the form submission and add the new contact using Model.add(Model_Contact contact).
- The value of an input field is accessible through field.value.

4.6.2 Listing contacts

Now have a look at Listing 15 – the event handler for listing all contacts.

Listing 15: list_handler event handler

```
1  // Event handler for listing all contacts.
2  var list_handler = (Event event) {
3
4    // Prevent default action of link similar to JQuery.
5    event.preventDefault();
6
7    // Get all contacts.
8    List<Model_Contact> contacts = Model.find_all();
9
10   View_List view = new View_List(contacts);
11
12   // Clean content area before appending the list:
13   document.query('#content').nodes.clear(); // Equivalent to JQuery's empty().
14
15   // Now add new list of contacts.
16   document.query('#content').nodes.add(view.element); // Equivalent to JQuery's
         append().
17 };
```

- First the handler gets all contacts through Model.find_all().
- Equivalent to the other view the content is cleared and the view appended to the content.

4.7 Result

The result could be described as responsive web application. The syntax and a few features of Dart could be shown and will give a brief overview of Dart's capabilities. Nevertheless it is worth having a look at the complete Dart language tour[9] and find out more about Dart.

[9] http://www.dartlang.org/docs/language-tour/

5 Evaluation

5.1 Development state

Sure, Dart is currently a technical preview. Thus certain aspects of the language will not work properly and some concepts – like abstract classes – are not implemented yet. The language may change throughout the development process and for stability there is still some work to do.

5.2 Browser support

For now no browser but Dartium does support Dart natively. Meaning developed Dart applications have to be compiled to Javascript so they will work with common used browsers.

Sure for a technical preview this is ok, but unfortunately "[as] of March 2012, Microsoft Internet Explorer, Mozilla Firefox, Opera Software's Opera browser, and Apple Safari do not have plans to implement support for Dart". [5]

5.3 Learned from Javascript

There are some things to point out Google learned from today's web development and especially from Javascript regarding client-side development: [3]

- Built-in solution for working with libraries in library scope – no global namespace.
- Powerful and easy-to-use HTML library based on CSS selectors (similar to JQuery).
- Class-based object-orientation supporting abstract classes and interfaces instead of prototypes.
- Comfortable type checking due to optional typing.

6 Related work

Due to the early state of Dart the list of related work is limited to only a few books and a handful active blogs.

Blogs like DartWatch[10], Dart Vader[11] or dartr[12] just like Seth Ladd's Blog[13] who has also co-authored "What is Dart?"[14] have been active since the early days of Dart.

Furthermore there are two books currently in progress: "Dart for Hipsters"[15] from Chris Strom[16] who is also developing a Dart-based MVC Framework "Hipster-MVC"[17] and "Dart in Action"[18] from Chris Buckett. They will cover both client- and server-side development with Dart in more detail.

[10] http://dartwatch.com/
[11] http://dartvader.grobmeier.de/
[12] http://dartr.com/
[13] http://blog.sethladd.com/
[14] http://shop.oreilly.com/product/0636920025887.do
[15] http://dart4hipsters.com/toc.html
[16] http://japhr.blogspot.de/search/label/dart
[17] https://github.com/eee-c/hipster-mvc
[18] http://www.manning.com/buckett/

7 Conclusion

For now Dart should not be used for production environments, but for broadening ones horizon web developers should have a look at Dart – have a look at the sample applications or try the playground[19].

Throughout the paper Dart has shown capabilities to develop responsive web applications. Beneath a familiar syntax Dart provides object-orientation, a great collection framework, optional typing and an easy-to-use HTML library.

The remaining question is: **Will Dart be supported natively by the popular browsers?** I am sure Dart could make a great job, but it is about waiting for statements from the big browser developers whether they think about implementing support or not.

[19] `http://www.dartlang.org/`

References

[1] Google. Technical overview. http://www.dartlang.org/docs/technical-overview/, 2012. Accessed: 30/04/2012.

[2] Google. A tour of the dart language. http://www.dartlang.org/language-tour/, 2012. Accessed: 30/04/2012.

[3] Christian Grobmeier. Dart: 10 punkte, in denen es javascript übertrifft. http://t3n.de/news/dart-10-punkte-denen-javascript-358345/), 2012. Accessed: 30/04/2012.

[4] Mark S. Miller. Future of javascript. https://gist.github.com/1208618, 2011. Accessed: 30/04/2012.

[5] Wikipedia. Dart (programming language). http://en.wikipedia.org/wiki/Dart_(programming_language), 2012. Accessed: 30/04/2012.