The BaCS GUI Framework
A Browser and Client-Standalone Application GUI library

Project description

Google SoC 2006 Application

by Alexander Stockinger
May 2006
stockina@fmi.uni-passau.de

1. The BaCS vision 3
2. Introducing BaCS' components 3
jSWT client library 3
RPC client & server 4
jSWT server library 4
SWT / jSWT wrapper 4
3. jSWT client library 4
Obstacles 4
Design 5
A simple example 5
4. The BaCS RPC library 6
The approach 6
Session handling 6
Design and portability 7
A simple example 7
Required bandwidth 7
5. jSWT server library 7
6. SWT / jSWT wrapper (Java only) 7
7. Performance and security considerations 7
Bandwidth 8
Latency 8
Server CPU load 8
Client side security 8
Server side security 8
8. License of BaCS 8
9. Appendix: Development approach 9
Phase 1: Requirements engineering 9
Phase 2: Design 9
Phase 3: Programming 9
Phase 4: Integration 9
Phase 5: Delivery 9
Phase 6: Maintenance 9
10. Appendix: Project state 9
11. Appendix: Project Schedule 10
12. Appendix: References 11

Web application development has always been a tedious task: lack of debugging support, browser incompatibilities and the problem of separating program code from HTML make programming such projects harder. While this sounds like a programmer's nightmare many developers have to bother with these problems on a daily basis. Even more concerning: many applications require both a company internal standalone application and a web interface for customers or partners. While ASP.NET and Java offer ways to reuse code in both worlds, UI programming has always been a double task. With the Web 2.0 hype gaining momentum this problem will become more and more obvious.

1. The BaCS vision

The BaCS project was initiated with the goal to solve at least a number of these tedious issues associated with web development today. BaCS is short for "**B**rowser **a**nd **C**lient-**S**tandalone" – an accurate description for the ultimate vision of the project: to allow software developers not to care anymore if they write browser based web-applications or client side standalone programs – a situation many developers have been dreaming about. Leveraging the potential of many powerful web technologies BaCS offers a way to make web development just as easy as writing a comparable desktop application.

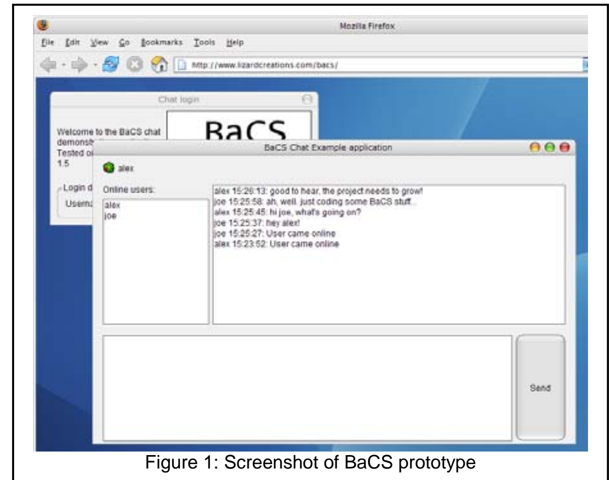


Figure 1: Screenshot of BaCS prototype

Where the WWW began with simple HTTP and HTML based communication, the need for dynamic content became obvious quite soon. CGI and eventually special server side programming environments like PHP, JSP and ASP.NET allowed for complex web applications but still the state-less page paradigm was untouched, once a page was loaded it could not be modified from the server anymore. Just recently a new approach using the existing technologies came up that allowed for changing the contents of the website using information from the server **after** the page was fully loaded: known now as AJAX.
BaCS takes the next step in the continual evolution: making extensive use of this exciting new AJAX technology the BaCS framework opens up the possibility to have a rich feature set and a previously unseen "application-feeling" inside a browser (apart from Java applets or Macromedia Flash programs perhaps – both requiring plug-ins to work) while entirely taking the burden off the developer to write a single line of HTML. The amazing simplicity of writing incredibly rich web applications with BaCS is only excelled by the development time savings emerging from the idea of not having to write two separate GUI front-ends in situations where both a standalone and a web application are required.

Using the BaCS library as a software platform can:

- Shorten your development time
- Significantly increase code reusability
- Open up a new level of web applications
- Easily develop extensive web projects
- Greatly enhance your users' experience

2. Introducing BaCS' components

To understand how BaCS can enhance your applications and software suites one has to know a little bit about the projects internals. BaCS is a bundle of different libraries that work together, providing a framework where the developer doesn't have to worry about the fact that he is developing web applications anymore. In the following section a brief overview of these libraries along with a short description is provided to convey the "big image" of BaCS' internals. Later a detailed description of each component will be presented.
In Figure 2 you see a graphical display of the communications between the modules in web based applications (PHP and Java in this case) compared to a client side application using SWT via BaCS.

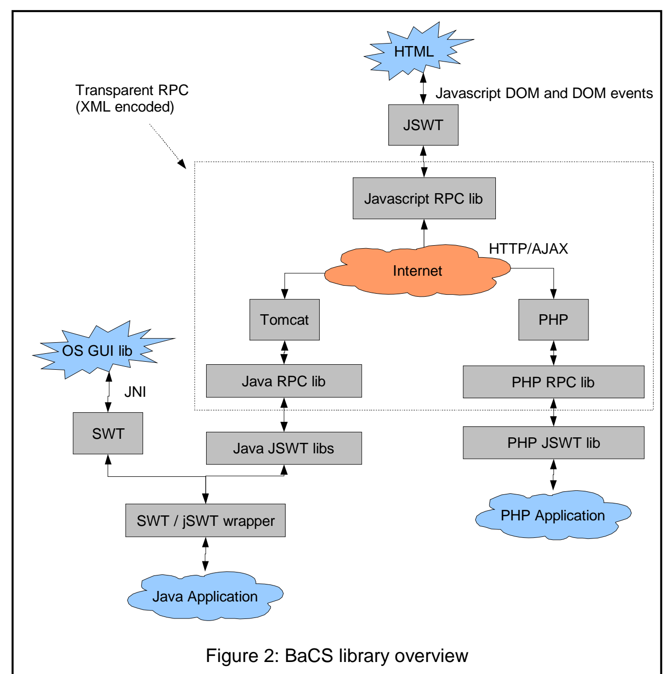


Figure 2: BaCS library overview

jSWT client library

The jSWT (short for **J**ava**S**cript **w**idget **t**oolkit) client library offers a SWT like API for programming web applications in JavaScript. SWT is the name of the Standard Widget Toolkit, developed by IBM for the Eclipse project. For

client-only web applications jSWT can be used as a standalone library as well.

RPC client & server

BaCS' RPC (**r**emote **p**rocedure **c**all) library is a simplified server-to-client RPC library allowing the server to execute dynamically generated JavaScript code in the browser. Its task is to bridge the communications gap between the client side GUI management and the server side application code.

jSWT server library

The jSWT server library is a translation library for transparently issuing client side GUI commands via the RPC library. Its task is to offer a SWT-like API to server applications and to entirely hide any details about client-server communication. Still it's a very thin and simple layer, again facilitating portability and maintainability.

SWT / jSWT wrapper

For the ultimate goal of writing an application once and then deploying it both as web project and standalone application the SWT/jSWT wrapper offers the required functionality to completely hide web-specific problems and issues and provide a simple GUI library for both worlds.

3. jSWT client library

The jSWT client library takes the burden of actually generating HTML and DOM code off to the client. The server does not have to write any HTML code itself. jSWT accomplishes this task by offering a SWT-like programming interface to the developer and then translating all visual GUI components to HTML using the W3C DOM interface present in all modern browsers. By implementing jSWT in JavaScript it was made sure that absolutely no additional software like plug-ins are required on the client side for running BaCS based web applications – eventually opening up even mobile internet platforms like cell phones once their browsers reach the functionality of current desktop browsers.

Obstacles

Of course a module like jSWT, using existing technologies invented years ago where the way the web was going could not be foreseen, has to face a number of obstacles. These have been analyzed, and for each of them a solution has to be found in order to provide jSWT's services that are so vital to BaCS.

Browser incompatibilities

A general problem with web development is browser incompatibility. While there are clear definitions of browser behaviour by the W3C especially the most used browser seems to have problems obeying them. Considering the fact that a standard can probably never be precise enough not to allow slightly different implementations the situation where all web browsers behave in the same way is unlikely. These issues are faced by calling wrapper from a small compatibility library where possible, implementing workarounds where necessary and using duplicate workaround code only where unavoidable. While it is possible to choose different code paths depending on the browser, jSWT tries to avoid this technique as much as possible in order to maintain a clean code base - even if this choice imposes performance issues in some browsers.

Memory management

JavaScript in browsers is usually not implemented with large applications in mind. General issues like memory management suddenly reach greatest importance once a JavaScript application contains more code than just a few lines to change the image of a link button when the mouse pointer hovers over it. An AJAX based application can run for hours, execute hundreds of thousands of lines of code and sadly enough memory management isn't that good in some widespread browsers. This problem has to be taken into strong consideration.

A message queue is required

JavaScript does not offer a `sleep()` command. Although one might occasionally wish for such a statement it makes perfect sense not to include it in a browser environment. For a widget toolkit however this can impose problems. Using self-layouting code (like SWT does) the library often needs to know about things the browser might not know, yet. A good example is an image in a label widget. jSWT can only layout the window if it knows the size of the image. Waiting for that image in a busy loop is not an option: browsers today detect that and will eventually terminate the script. Preloading the image might be an option for small applications, but for large application suites it is obviously not feasible to preload every single file that might be relevant at some later time.

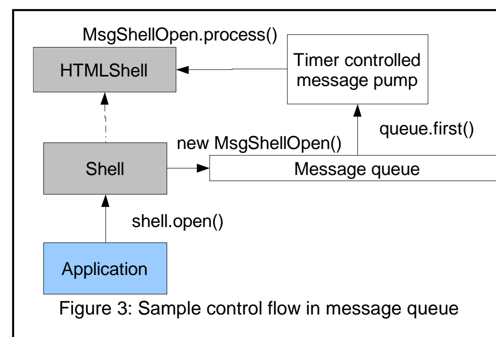


Figure 3: Sample control flow in message queue

The solution used by jSWT is taken from existing GUI libraries - all actions go through a message queue. In the case of jSWT a little addition was made: message can state that they are not finished, yet. In the previous example this mechanism is used to keep the message in the queue until the image has finished loading. Only when this action has completed the next message (e.g. a message that performs a window layout) can be processed. The message queue is maintained by a timer that frequently checks the state of the current message while leaving enough CPU time for the browser itself and does not require any sort of waiting anymore.

Full page applications

While jSWT usually is using windows, a well established interface metaphor known to any user, for interaction, some web-only applications will want to stick to the more web-like full page style. To accommodate this requirement jSWT introduces a construct not known in the original SWT: the desktop. The web developer marks a node in the DOM tree as desktop node and jSWT offers functionality to place widgets directly into it – without the need for a window. To allow for a great freedom in content design an `HTMLLabel` widget is implemented – offering a sandbox environment injected into the jSWT widget system that allows for simple HTML nicely embedded in jSWT applications.

Design

Class layout

The class hierarchy with all interfaces in jSWT is taken from SWT. However these classes form only a thin layer that maintains information for the widgets and manages message dispatching on GUI relevant calls. All drawing operations are performed in separate classes, keeping away DOM implementation details from the actual functionality of the code – a concept well known in software engineering as the “Bridge Pattern”. While clean code is a clear advantage of this approach the structure was also made almost necessary by the usage of a message queue.

Interface

The programming interface of jSWT is clearly taken from SWT. Some changes were made (e.g. no listener classes but callback functions), but the overall way of programming jSWT strongly resembles SWT. This is an obvious advantage to programmers already used to SWT but also allows developers new to web application programming a smooth experience during their first experiments since SWT is easy to learn and well documented.

Theme support

Web design and web programming usually involves the requirement to stick to a specific look. Whether it’s a corporate identity with shapes, colours and fonts that has to be obeyed or it’s an already existing web presence with a unique design where a newly integrated web application must ensure that no breach in the user experience occurs.
jSWT faces this problem by strictly separating interface, appearance and functionality in distinct classes. A base class `Theme` exists, providing the interface a new Theme has to implement. A new widget theme is thus implemented by simply creating a new subclass of `Theme` – overriding a number of functions to create and manipulate a DOM hierarchy.

A simple example

```
// Create the shell
var shell = new Shell(jswtGetDesktop(), SWT.DIALOG_TRIM);
shell.setText("Chat login");

// Create welcome text
var label = new Label(shell, 0);
label.setText(cWelcomeText);
var gridData = new GridData();
gridData.grabExcessHorizontalSpace = true;
gridData.widthHint = 150;
label.setLayoutData(gridData);

// Create logo
var label = new Label(shell, 0);
label.setImage("chatlogo.gif");

// Create login widgets group
var group = new Group(shell, 0);
group.setText("Login data");
group.setLayout(new GridLayout(3, false));
var gridData = new GridData();
gridData.grabExcessHorizontalSpace = true;
gridData.horizontalAlignment = SWT.FILL;
gridData.horizontalSpan = 2;
```

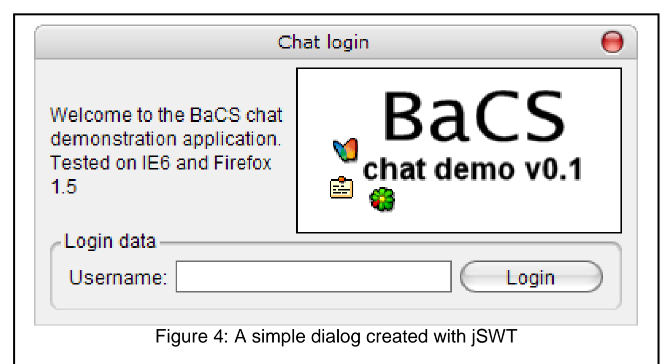


Figure 4: A simple dialog created with jSWT

```
group.setLayoutData(gridData);

// Create the “Username:” label
var label = new Label(group, 0);
label.setText("Username:");

// Create the username text
// input field
var username = new Text(group, 0);
var gridData = new GridData();
gridData.grabExcessHorizontalSpace = true;
gridData.horizontalAlignment = SWT.FILL;
username.setLayoutData(gridData);

// Create the ok button
var ok = new Button(group, 0);
ok.setText("Login");

// Set layout for shell, pack it to desired size and open it
shell.setLayout(new GridLayout(2, false));
shell.pack();
shell.open();
```

As you can see, the BaCS way of writing web application interfaces has little to nothing in common with the traditional HTML based approach. Programmers knowing SWT will immediately see the strong resemblance to SWT based standalone applications written in Java. The library takes a widget hierarchy, layout hints and widget commands and directly translates them to HTML code using the W3C Document Object Model – all nicely hidden inside the framework, far away from the programmer.

4. The BaCS RPC library

Having jSWT providing solid and well implemented client side GUI services in a browser it's now required to have a mechanism for handing control of the user interface over to the server. While this may at first seem simple – nothing more than sending a web page containing some server-side generated JavaScript code – this turns out to be a tricky part. In BaCS no page-loads happen at all once the first HTML file is displayed, so all communication between client an server has to be handled differently. With AJAX the required functionality is already available: a way to exchange information between the two hosts even after the page has finished loading. Leveraging this technology, the RPC library is responsible for all network communication in BaCS.

The approach

The traditional way RPC is used usually places the server in the role of the service provider, answering procedure call requests by the client. The BaCS RPC library reverses the roles. It's not the client that requests information from the server; it's the server telling the client what to do. This idea is the logical conclusion from the fact that the browser merely acts as a GUI platform, displaying applications running at the server.

The RPC library is split into two components: the server library and the client library. The server library offers a extensive set of constructs such as assignments, function calls, member function calls, array access and more. Those constructs directly translate to JavaScript language elements, offering a powerful way to express client side code in a language independent way. Language independency is accomplished by using XML as the intermediate language – all RPC calls are translated to a simple and user readable XML structure that is then interpreted and translated into JavaScript code by the RPC client library. The decision to use XML was an easy one considering its advantages: excellent support on all platforms and languages, clean Unicode character handling, strong advantages during development (user readable), extensions are easy to integrate and handle.
The client library implements the necessary translation code required to execute the RPC requests encoded in the XML stream sent by the server. However the abilities to execute server side code from the client library is very limited. There is no need for extensive client-to-server RPC functionality – keep in mind that in BaCS the browser only acts as a GUI service provider. All client-to-server communication required consists of event notification and polling for GUI changes.

Session handling

In many server side languages every single request is processed in a separate instance – bringing all the disadvantages of a state-less application with it. In most such environments this problem is addressed by offering a session management system, allowing keeping information assigned to one client across multiple requests. BaCS requires such functionality, and the BaCS RPC library handles the task of keeping track of session organisation – after all languages like PHP depend on techniques like URL rewriting to convey session information across page loads – a method that will not work with requests dynamically generated at the client side. However this problem is easy to cope with and a solution was found that does not require cookies to be stored at the client side.

Design and portability

The BaCS RPC library generally consists of a small XML encoder / decoder at both the client and the server side. Additionally a request-queue was implemented to ensure mutual exclusion of server-controlled actions in the GUI. The design of the library strongly reminds of a recursive descent parser at the decoder side – obviously a good choice since parsing a XML structure is the job to be accomplished. The encoder on the contrary reminds of a somewhat simple compiler – translating server side function calls to XML encoded RPC information.

For portability only the server side library has to be considered. The BaCS RPC library is small, lightweight and simple. Porting the library to other languages is a trivial task provided a XML library is available.

A simple example

The XML code used in the BaCS RPC library is simple, the structure strongly reminds of a syntax tree in a compiler backend. A short example directly taken from a server-to-client packet demonstrates its look and feel:

```
<rpc sessid="9cc84fceda2986a431af931043e62654">
  <call fname="debug">
    <object type="constant">
      <constant type="string">Hello world</constant>
    </object>
  </call>
</rpc>
```

This XML code directly translates into the following JavaScript code:

```
debug("Hello world");
```

Required bandwidth

The RPC module is the bandwidth intensive part of BaCS – it handles real-time updating of the GUI and event notification of the server, so data is constantly sent back and forth between the server and the client. All other components either don't impose internet traffic at all, or require static resources that can be loaded once and then fetched from cache on demand.
Compared to binary data encoding XML always imposes an extra overhead for structural elements that are not required for the application's understanding of the data. This issue can simply be addressed by transmitting the data gzip compressed, a HTTP 1.1 feature available in virtually all modern browsers. A closer look at possible bandwidth problems is taken in the chapter "Performance and security considerations" later in this document.

5. jSWT server library

Having jSWT and the RPC library in place, the next task to accomplish is the offer the comfortable SWT interface available at the client to the developer writing the server side application. This job is performed by the jSWT server library by translating calls the programming interface into RPC commands using the BaCS RPC library. With the two previous components in place this is a very easy job – consequently the jSWT server library is merely a thin layer between the specific application and the RPC component.
Keeping this fact in mind the drawback of requiring a language dependant implantation of this server library (e.g. in PHP or Java) becomes less problematic. A port to another server platform is a rather trivial task to accomplish once the RPC library is available as well. Some constructs like a callback mechanism have to be implemented with a language's specific feature set in mind, but an experienced programmer should have no problem finding a well established way to solve these problems in any OOP based environment.

6. SWT / jSWT wrapper (Java only)

For a Java based server environment (e.g. Tomcat) there's an extra feature: Java is the origin of the original SWT library implemented by IBM for the Eclipse platform. The close affinity between SWT and jSWT allows for a common programming interface for the two GUI libraries. A very thin layer of code extensively using aggregation can abstract from the differences and should enable developers to write applications once that then can be deployed both as standalone application and as a web service. This has not been implemented in a prototype though, so it's not yet clear what obstacles are to be expected.

7. Performance and security considerations

Reading expressions like "control handed to server" and "constant client-server communication" immediately concerns the performance-sensitive programmer. However the performance impact of BaCS compared to traditional web applications scales fairly well to the enhanced user experience and functionality as you will see in the next paragraphs. Security issues have to be considered as well– a requirement inherent to web applications since they send possibly sensitive data all around the planet over an open network.

Bandwidth

As stated earlier most transmitted data is XML encoded RPC messages which like most human readable text based formats are easily compressible and can thus be greatly reduced in size. Considering that no loading of entirely new pages occurs in a BaCS application, the compressed XML data does in fact require less bandwidth than the traditional approach building on the state-less HTTP protocol. However the constant updating of the GUI performed by the browser of course imposes non-trivial bandwidth requirements - a price that has to be paid for many Web 2.0 applications (see Google Maps with vast amounts of satellite and map data).

Latency

Performing control flow entirely on the server naturally imposes latency not present in standalone applications – reactions to user interaction take their way from the client to the server, are processed there, and the actions to be taken are then sent back to the client in XML encoded RPC packages. The prototype however shows that with a sufficiently good connection at the client side (an end-user DSL connection has been used for testing) and a lower-cost server (10MBit/s connection used for test server) already fulfil the requirements for almost instant reaction to user input.

Server CPU load

Server CPU load is the last performance critical topic that has to be considered. With constant update requests from clients, even idle client applications impose server load. To keep server requirements to a minimum very short code paths are used for requests that are answered with no GUI changes to be performed.
Constant update requests from the client are only necessary to accommodate situations where GUI changes occur without user interaction. A way to reduce unnecessary update requests would be identify such situations on the server side and only then have the client send update requests constantly. This approach however is not planned for the current iteration of BaCS. Once the project is adopted by the community this topic will surely become a priority issue and will be addressed accordingly.

Client side security

Transmitting event data to the server and RPC code back to the client in an unencrypted format like XML on every user interaction doesn't sound like a very good method to ensure a user's privacy. But keep in mind that BaCS is building up on the HTTP protocol – for sensitive data the usage of HTTPS instead will immediately solve these problems and provide for a method where no sensitive data can be extracted from the hosts' communication.

Server side security

Since only events are transferred from the client to the server the security critical code section in BaCS' server modules is pretty small. A careful implementation will eliminate most security vulnerabilities on the server side. Of course the BaCS library cannot protect the application from processing information possibly dangerous in a specific context (think of SQL queries for example) so proper precautions have to be taken there. However this is a problem common to desktop applications and traditional web projects as well so no disadvantage or security issue is introduced by the usage of the BaCS library here.

8. License of BaCS

The license choice for BaCS is not entirely free. Since at least in the jSWT library code from the Eclipse project is used the Eclipse public license is imperative for jSWT. For the other parts of BaCS the license choice is not fixed, by having BaCS released under a single license would provide for clarity and acceptance in the community. However, with the exception of jSWT a mentoring organisation can also recommend more suitable license terms.

9. Appendix: Development approach

To analyze the necessary steps in the development of BaCS the waterfall model for software engineering shall be used as archetype. While this is a rather unconventional approach for an open source project, using that model as a guideline has its advantages: it assures a level of quality, allows for structured development and offers hints about where to look for caveats and how to avoid them.

Phase 1: Requirements engineering

This phase usually includes writing a requirements specification document. Since the requirements for a GUI library are usually clear and an already finished GUI library is taken as example this phase can be cut short for the development of jSWT. The other components are described sufficiently in this document, so this introduction to BaCS together with the SWT javadoc can be seen as a requirements specification document, all describe functionality shall be implemented during Google Summer of Code 2006.

Phase 2: Design

The basic design is already laid out in this document; more detailed design considerations would be required for team development. Since Google SoC is aiming at one-person projects this step can be skipped. Also there is already very detailed information on SWT, so no further work in this phase is required.

Phase 3: Programming

This phase will take the mayor part during the time of Google SoC 2006. A schedule for this phase can be found in the appendix “Project Schedule”. This phase also includes detailed documentation of the modules and their code. The result of this phase will be usable and tested software modules which are specified earlier in this document.

Phase 4: Integration

Along with Phase 3 goes module integration. In this step the existing models are put together to a larger piece of software. Since BaCS is itself just a library, this phase will be realised by writing demonstration applications that then can also serve as test projects for BaCS.

Phase 5: Delivery

This is the phase that is basically the goal of a Google SoC mentored project: a finished product, ready to ship. Evaluation is not uncommon during this phase, which will be done by the mentoring organisation. Delivery is then due at the end of Google SoC 2006.

Phase 6: Maintenance

After the project was handed off to the mentor and the evaluation is finished, the project shall then be officially released in a stable state. Downloading the source code will then be possible via a revision control system over the internet, documentation will be available online, a bug report system has to be installed and a project wiki is planned.
Maintenance of the project will then hopefully be supported by a community of users of the BaCS project, taking successful open source projects as an example.

10. Appendix: Project state

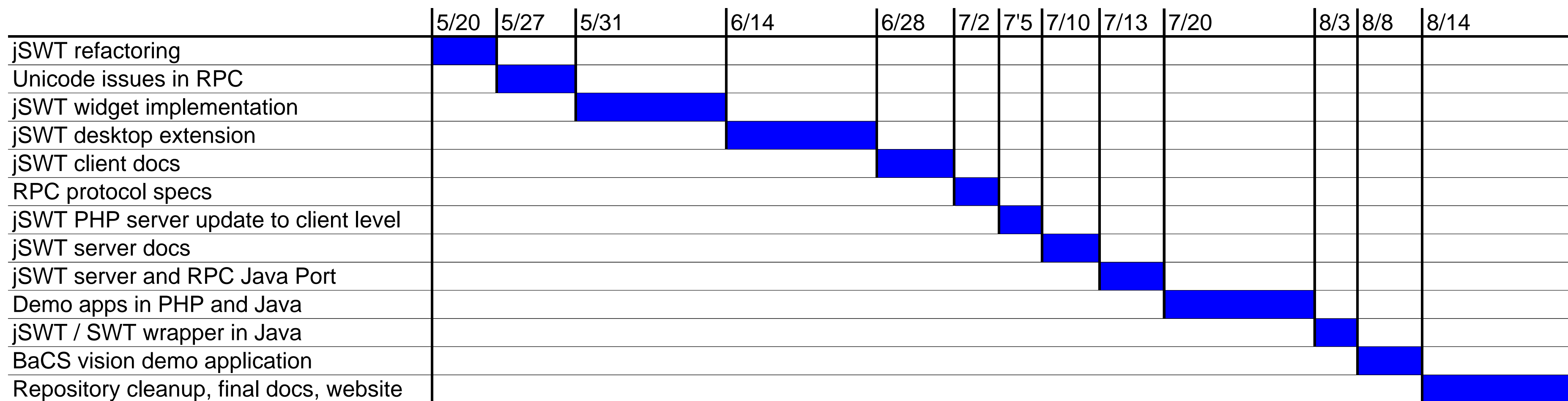
Development of BaCS has started in late April 2006, about 10 days before the student’s sign up process for the Google Summer of Code 2006 began. A prototype with proof-of-concept character exists and is available for consideration at http://www.lizardcreations.com/bacs. The following functionality is already implemented:

- jSWT client library: basic shell, button, text and label are implemented, first event handling, message queue
- BaCS RPC: XML to JavaScript translation implemented for assignments, calls, member-function-calls, array access
- jSWT server library: small part of jSWT to XML translation functions implemented in PHP

To demonstrate the proposed concept and its potential a simple BaCS based chat application has been implemented entirely in PHP, using mySQL to store data. Development of the entire chat application took very little time and code and strongly reminded of writing a standalone application, underlining the simplicity of creating web applications using the BaCS library as a framework.

11. Appendix: Project Schedule

To complete the project within the 3 months timeframe a tight schedule is required. Since Google SoC 2006 greatly overlaps with the time where university is in session at the University of Passau I can only invest roughly 15 to 20 hours per week in average over the whole time: while university is still in session the curriculum allows for around 10 to 15 hours per week, afterwards around 20 to 25 hours per week. Since I have other obligations next to university (as can be seen in my resume) a maximum of 30 hours per week for the project has to be considered.
According to this information I have created the following schedule for the development of BaCS during the time of Google SoC 2006:



12. Appendix: References

Google Summer of Code website: http://code.google.com/soc
BaCS demonstration website: http://www.lizardcreations.com/bacs
SWT website: http://www.eclipse.org/swt/
W3C DOM website: http://www.w3.org/DOM/
W3C XML website: http://www.w3.org/XML/
Wikipedia introduction on AJAX: http://en.wikipedia.org/wiki/AJAX
Explanation of the waterfall model for SE: http://courses.cs.vt.edu/csonline/SE/Lessons/Waterfall/index.html
Zoho-writer AJAX application: http://www.zohowriter.com/Home.do
Google Maps AJAX application: http://maps.google.com