CS 461 / 462 / 463
Client Requirements Document

Task Scheduler API for Firefox OS

Team Name: FxOSU

Team Members:
John Zeller (zellerjo@onid.oregonstate.edu)
Jonathan McNeil (mcneilj@onid.oregonstate.edu)
Pok Yan Tjiam (tjiamp@onid.oregonstate.edu)

Client/Sponsor/Mentor:
Dietrich Ayala
Organization: Mozilla
Phone: 503-512-9252
E-mail: dietrich@mozilla.com

Introduction to the problem:
Firefox OS is an open source operating system for smartphones developed by Mozilla. It is a web-centric system based on Linux kernel. Firefox OS phones have launched in 15 countries around the world. It targets low cost market in developing countries.

Project Description:
Developing countries like India usually have poor quality infrastructures. Unreliable network and unreliable electrical grid is a huge problem that affect user experience. The poor network condition makes some of the applications become non-functional. Charging the battery of the phone can even take days for users living among the most unreliable electrical grids.

A solution to this problem is a job scheduler API which could allow the phone to delay a task until it has a good network connection or has a power cord plugged in. This should preserve the functionality of the phones under different network condition and reduce inefficient use of network access. Since all Firefox products share the same core, this API is expected to be deployable in not only the Firefox OS but also in Firefox for Android and Firefox for Desktop.

Requirements:

Req. #	Requirement	Status	Comments
1	Analyze existing efforts (ie ServiceWorkers, RequestSync) to determine if they can be integrated with our system.	Pending	
2	The API should integrate with existing efforts wherever possible.	Pending	
3	The prototype API should be written in JavaScript	Pending	
4	The API should be written in C++	Pending	
5	The API should be callable by JavaScript executing in the DOM in a web context.	Pending	
6	The prototype API should be able to queue tasks	Pending	
7	The prototype API should be able to prioritize tasks in the queue based on network, battery and system data.	Pending	
8	The prototype API should be developer configurable, to provide a force execution of a task after a period of time	Pending	
9	The prototype API should be able to function without error on Firefox OS, Firefox for Android, and Firefox for Desktop	Pending	
10	The API should be able to access data on the quality of the network in order to determine if a task should be executed.	Pending	
11	The API should be able to access data on the charging state of the device in order to determine if a task should be executed.	Pending	
12	The API should be able to access data on the battery level of the device in order to determine if a task should be executed.	Pending	

13	The API should be able to access data about system load on the device in order to determine if the device can handle another task.	Pending	
14	The API should passively collect network status information.	Pending	
15	The API should analyze network data to determine patterns in network reliability	Pending	
16	The API should be able to queue and postpone tasks from being dispatched	Pending	
17	The API should be able to prioritize queued tasks based on accessible data about the device and the environment	Pending	
18	The API should have a mechanism for dispatching queued tasks	Pending	
19	The API should be developer configurable, to prioritize application specific tasks	Pending	
20	The API should be developer configurable, to provide a force execution of a task after a period of time	Pending	
21	The API should be able to function without error on Firefox OS, Firefox for Android, and Firefox for Desktop	Pending	
22	The test app for the API should be written to function on Firefox OS, Firefox for Android, and Firefox for Desktop	Pending	
23	The test app for the API should benchmark device resource usage, contrasting use with the API and use without the API	Pending	
24	Write a Web IDL specification for the API implementation	Pending	

Versions:

- Version 1: Create a prototype in Javascript. It does not need to be fully functional.
    - 1.0 Network monitoring functionality
    - 1.1 Process and Power monitoring functionality
    - 1.2 Task Queuing functionality
    - 1.3 Task Prioritization functionality
- Version 2: Implement the components of job scheduler API in C++.
    - 2.0: Network, Power and Process monitoring functionality
    - 2.1: Fully functional job scheduler with task queueing and dispatch functionality
- Version 3: Combine all functionalities into the final implementation of the job scheduler with testing application included.

Design:

This job scheduler API will be put into the network stack of Firefox OS which is called Necko. Necko will be where we obtain all of the necessary data to determine whether a task should be performed. We will then "schedule" tasks based on this information in a prioritized queue. These tasks will then be dispatched when the conditions are appropriate or a predetermined amount of time has passed (we don't want starvation).

The major components of this API will be systems to monitor network quality and traffic, charging state and battery life, and current system load. There will also be a scheduling system which will place tasks in the queue to be dispatched when the conditions are correct. There will also be a system which prioritizes delayed to prevent overload when good network conditions or the charging state of the device changes.

Specific tasks to be undertaken:

- Network monitoring function
- Power monitoring function
- Process monitoring function
- Developer configuration options
- Task queueing function
- Task dispatch function
- Testing application
- Draft a Web IDL specification document

Risk Assessment:

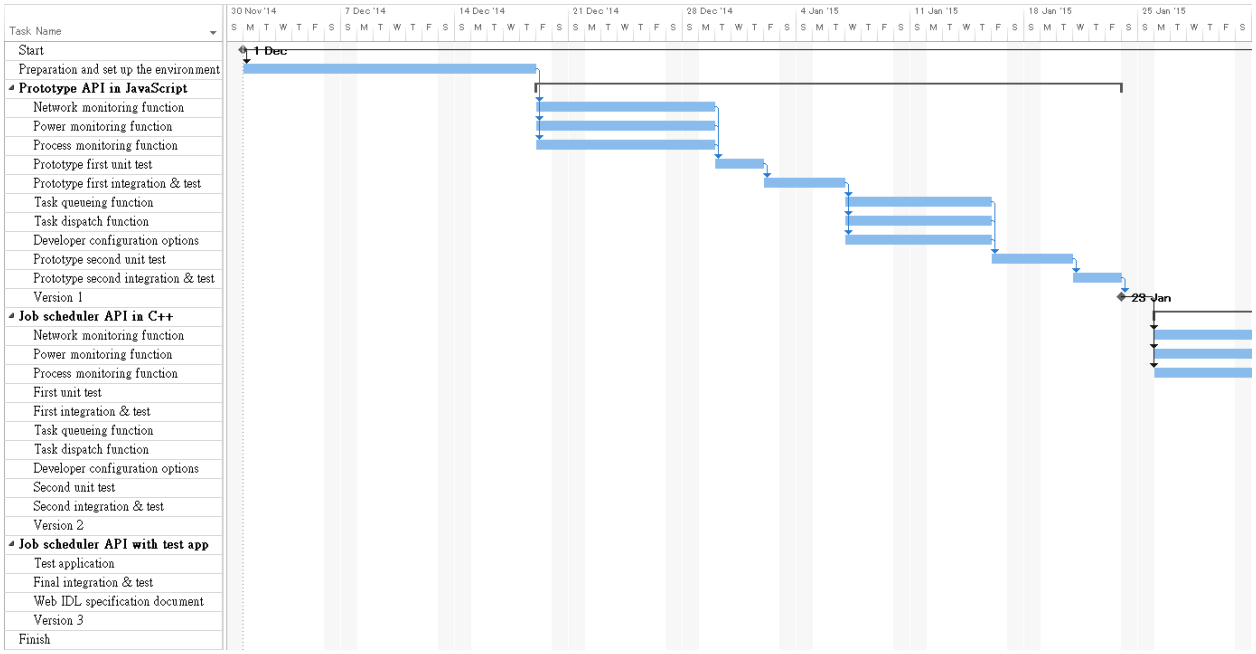
One of the challenges associated with creating an Programming Interface that is designed to minimize power usage is creating the interface in a way that doesn't hurt more than it helps. It could be useful if we could create a functionality that can schedule tasks, but if our method of doing so costs more battery usage then it saves we really haven't solved the problem. This applies to all of the monitoring functionalities which we intend to develop. If the constant monitoring of the devices resources is too costly I fear we may not actually solve the problem, but only provide a useful functionality for developers to use as part of the solution to the problem. This gets even more difficult as we might not even see it happening. We won't be able to tell if our API has real world results until it has been tested.

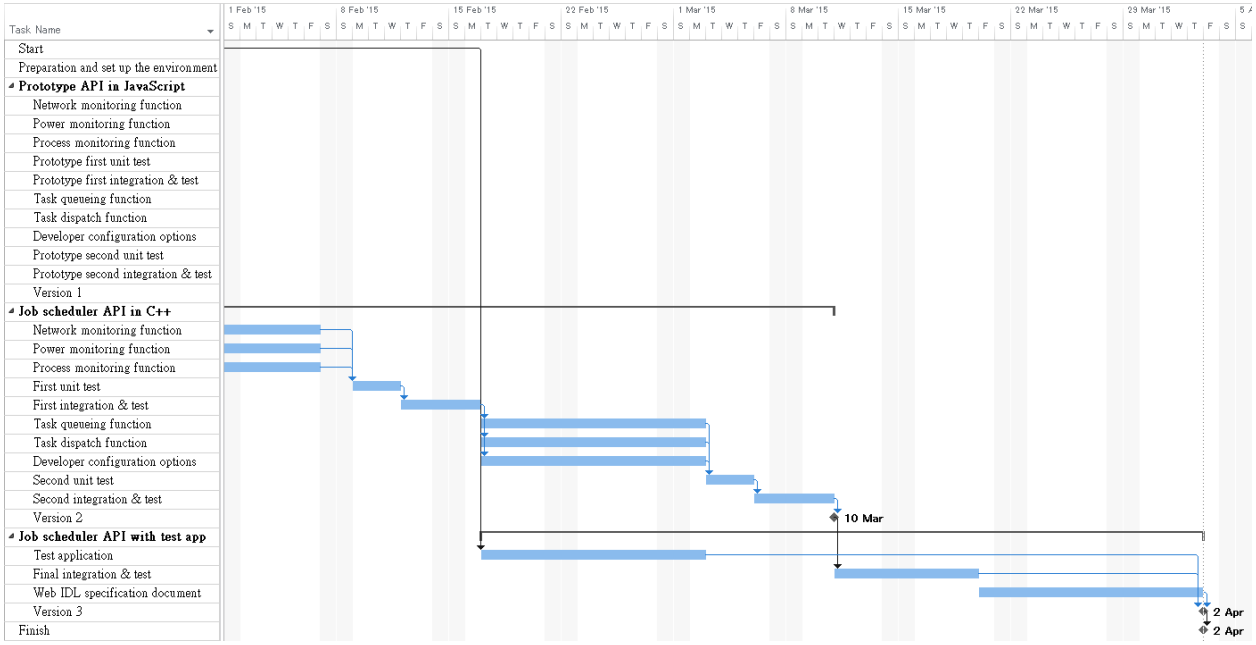
For any of the challenges associated with this project, both foreseen and unforeseen, there are a couple of options for what to do. Development in the mozilla core is not something that is being done alone. There is a thriving IRC network with developers with far more experience than any of us. The first thing I would do is start a discussion here to see if there are any methods of resolving the issue. Even if a direct solution doesn't present itself, we may be put into contact with someone who can give us more information in the area where we are experiencing the challenge. If a challenge becomes a disaster, the final option would be to sit down with our client and talk about work-arounds or requirement changes. This would be a last resort scenario.

Testing:

Unit tests will be created to test the functionality of the network, power and process monitoring functions and task queueing and dispatch functions separately. After all tests passed and all components integrated together, an integration test will be performed by building an example Firefox OS application to monitor the network and power usage of a Firefox OS device. The application will be used to test the performance of the Firefox OS device under different network conditions and battery charging states, and visualize the effect of the job scheduler in term of efficiency. We will perform the tests before each version released.

Preliminary Timetable:





Roles of the different team members:
The entire team will work together on the test application and Web IDL specification document. For the JavaScript prototype (version 1) and the scheduler API in C++ (version 2), we will divide the API into 6 functionalities in both versions and each of the member is in charge of 2 functionalities. Each member will have equal workload.

Integration Plan:
After finished the unit testing on each components and make sure they are all functional, we will integrate them into a single job scheduler API. We will upload our files to Github so that all the member can work together online. After the integration of all parts, we will do an integration test to see if our API works correctly or not.

References:

- https://developer.mozilla.org/en-US/Firefox_OS

Glossary:

- Gecko: a layout engine designed to support open internet standards like HTML
- Necko: the network library that provides an extensible API for several layers of networking

Signatures

Team members:

John Zeller	Date
Jonathan McNeil	Date
Pok Yan Tjiam	Date

Client/Sponsor/Mentor:

Dietrich Ayala	Date