|  |
| --- |
| HCL Technologies Ltd. |
| OpenXC - MOTECH Integration– Solution approach |
|  |

****

# Overview

OpenXC integration with MOTECH targets to integrate vehicle data and vehicle GPS location (taken from smartphone) on MOTECH framework.

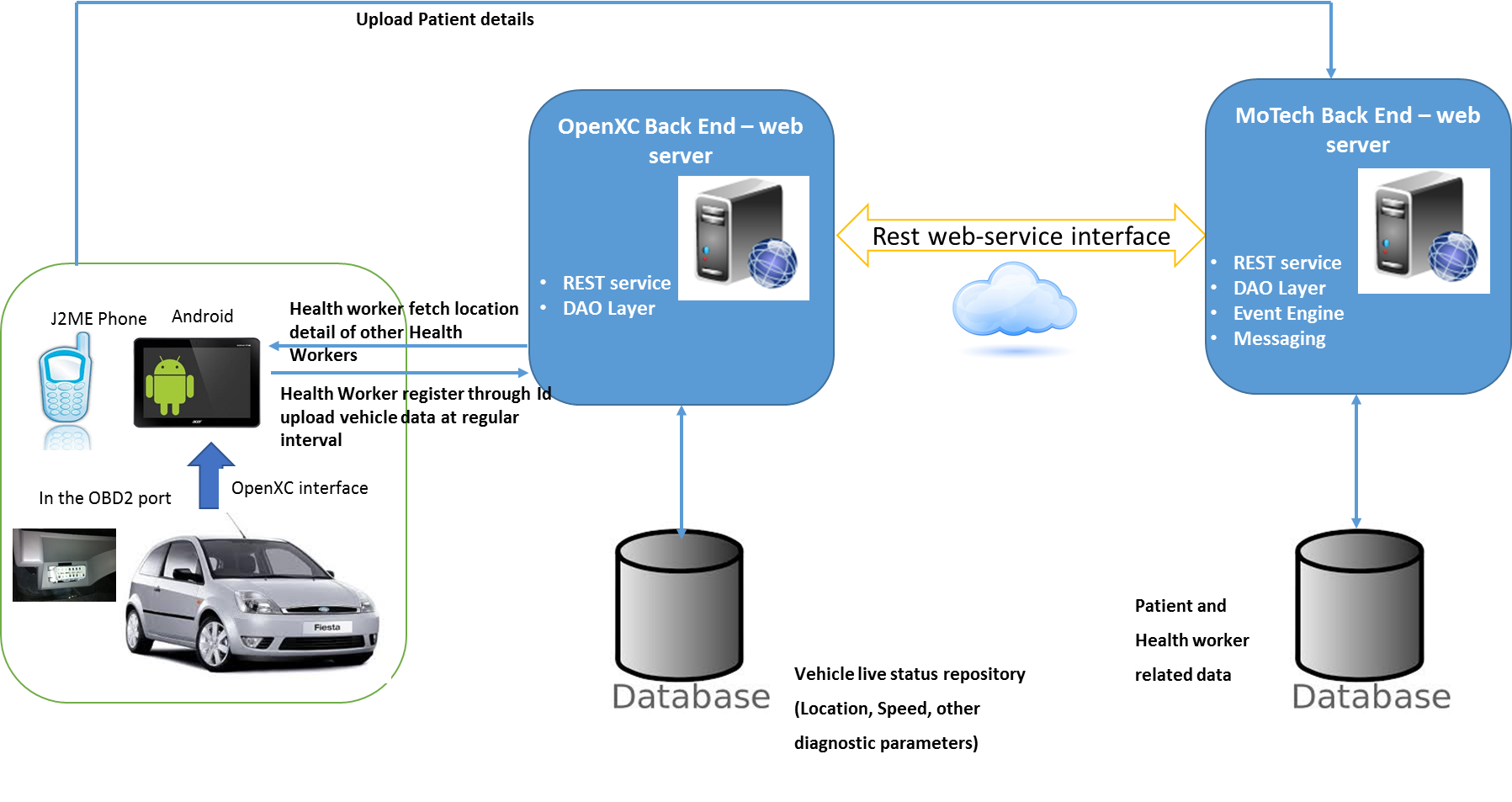
The doc represents a technical assessment and possible solutions for integration both from technical and functional perspective.

This will lead to **location based co-ordination** among health workers on their job in field.

This solution will be meant more for health workers/nurses and not for health recipients.

# Architecture Overview

A pictorial diagram on overview of integration solution –



**Highlights on above architecture**

* Health workers will be having two phones while travelling in-vehicle - J2ME device and Android device
* Both the devices will connect and interact with their respective backend – the most likely mode of connection will be cellular network
* J2ME phone connects to MOTECH backend
* Car are connected with OBD2 port and send parameter to Android device
* Android device connects with OpenXC backed

## Core solution approach

Integration achieved from the backend!

* Separate server for OpenXC

Basis for having a separate server for OpenXC –

1. By application point of view OpenXC and Motech are two distinct applications so need to prefer separate server for both
2. Application Isolation: You deploy an independent application to each server instance. Each server instance has separate settings and, because each server instance runs in its own Java Virtual Machine (JVM) or other, problems that one application encounter have no effect on other applications.
3. Load balancing: means no load on a single server.
4. Improves database read performance by spreading reads, if separate database server are used
5. Scalability: Enables horizontal scaling, i.e. environment capacity can be scaled by adding more servers to it.

## OpenXC Backend API Interface

OpenXC backend will facilitate in uploading vehicle data from each vehicle at a regular interval. And also that, required status could be fetched from any client device. There will be interface to notify message to all health workers.

**Some possible set of APIs are listed below:**

1. RegisterVehicle()

|  |  |  |  |
| --- | --- | --- | --- |
| **RegisterVehicle** | | | |
| **Protocol** | HTTPS - REST/SOAP API | | |
| **Description** | Send data to the OpenXC platform over https REST API. | | |
| **Input Parameter** | Data Format : JSON data | ***vehicleID*** | : String |
| ***vehicleType*** | : String |
| **Output** | Success/Failiure | ***Response Code*** | Success = 0, Failure = 1 |

1. UploadVehicleData()

|  |  |  |  |
| --- | --- | --- | --- |
| **UploadVehicleData** | | | |
| **Protocol** | HTTPS - REST/SOAP API | | |
| **Description** | Send vehicle data to the OpenXC platform over https REST API. | | |
| **Input Parameter** | Data Format : JSON data | ***vehicleID*** | : String |
| ***vehicleSpeed*** | : String |
| ***latitude*** | : String |
| ***longitude*** | : String |
| ***timestamp*** | : String |
| **Output** | Success/Failiure | ***Response Code*** | Success = 0, Failure = 1 |

1. GetCurrentLocation()

|  |  |  |  |
| --- | --- | --- | --- |
| **GetCurrentLocation** | | | |
| **Protocol** | HTTPS - REST/SOAP API | | |
| **Description** | Get data from OpenXC platform over https REST API. | | |
| **Input Parameter** | Data Format : JSON data | ***vehicleID*** | : String |
| ***timestamp*** | : String |
| **Output** | Data Format : XML/JSON | ***latitude*** | **:** String |
| ***longitude*** | **:** String |

1. GetAllVechicleCurrentLocation()

|  |  |  |  |
| --- | --- | --- | --- |
| **GetAllVechicleCurrentLocation** | | | |
| **Protocol** | HTTPS - REST/SOAP API | | |
| **Description** | Get data from OpenXC platform over https REST API. | | |
| **Input Parameter** | Data Format : JSON data | ***timestamp*** | : String |
| **Output** | Data Format : XML/JSON | ***latitude*** | **:** String |
| ***longitude*** | **:** String |
| ***vehicleID*** | **:** String |

1. NotifyAllwithmsg( )

|  |  |  |  |
| --- | --- | --- | --- |
| **NotifyAllWithMsg** | | | |
| **Protocol** | HTTPS - REST/SOAP API | | |
| **Description** | Send data to the OpenXC platform over https REST API. | | |
| **Input Parameter** | Data Format : JSON data | ***message*** | : String |
| **Output** | Success/Failiure | ***Response Code*** | Success = 0, Failure = 1 |

1. ShowRoute()

|  |  |  |  |
| --- | --- | --- | --- |
| **ShowRoute** | | | |
| **Protocol** | HTTPS - REST/SOAP API | | |
| **Description** | Send data to the OpenXC platform over https REST API. | | |
| **Input Parameter** | Data Format : JSON data | ***vehicleID*** | : String |
| ***latitude*** | : String |
| ***longitude*** | : String |
| **Output** | Success/Failiure | ***Response Code*** | Success = 0, Failure = 1 |

## APIs between OpenXC server and MOTECH server

For health workers records and validation

OpenXC server and MOTECH Server need to communicate with each other mostly for health worker’s verification and detail. Whenever health worker access OpenXC server, the verification of health worker with the MOTECH server is needed. Similarly, after tracing the current location of any health worker, if the system wants the detail of the health worker for further communication, then also OpenXC needs to communicate with MOTECH server to get those details.

Possible set of APIs are listed below:

1. verifyHealthWorker()

|  |  |  |  |
| --- | --- | --- | --- |
| **verifyHealthWorker** | | | |
| **Protocol** | HTTPS - REST/SOAP API | | |
| **Description** | Send data to the MOTECH platform over https REST API. | | |
| **Input Parameter** | Data Format : JSON data | ***healthWorkerID*** | : String |
| **Output** | Success/Failiure | ***Response Code*** | Success = 0, Failure = 1 |

1. getHealthWorkerDetail()

|  |  |  |  |
| --- | --- | --- | --- |
| **getHealthWorkerDetail** | | | |
| **Protocol** | HTTPS - REST/SOAP API | | |
| **Description** | Get data from MOTECH platform over https REST API. | | |
| **Input Parameter** | Data Format : JSON data | ***healthWorkerID*** | : String |
| **Output** | Data Format : JSON data | ***healthWorkerName*** | : String |
|  |  | ***contactNo*** | : String |
|  |  | ***Response Code*** | Success = 0, Failure = 1 |

## OpenXC Vehicle Data Stream

The present overall list of vehicle parameters that OpenXC provides is as below –

These signal names are a part of the OpenXC specification, although some manufacturers may support custom message names.

|  |  |
| --- | --- |
| **Parameters** | **Values** |
| steering\_wheel\_angle | * numerical, -600 to +600 degrees * 10Hz |
| torque\_at\_transmission | * numerical, -500 to 1500 Nm * 10Hz |
| engine\_speed | * numerical, 0 to 16382 RPM * 10Hz |
| vehicle\_speed | * numerical, 0 to 655 km/h (this will be positive even if going in reverse as it's not a velocity, although you can use the gear status to figure out direction) * 10Hz |
| accelerator\_pedal\_position | * percentage * 10Hz |
| parking\_brake\_status | * boolean, (true == brake engaged) * 1Hz, but sent immediately on change |
| transmission\_gear\_position | * states: first, second, third, fourth, fifth, sixth, seventh, eighth, ninth, tenth, reverse, neutral * 1Hz, but sent immediately on change |
| gear\_lever\_position | * states: neutral, park, reverse, drive, sport, low, first, second, third, fourth, fifth, sixth, seventh, eighth, ninth, tenth * 1Hz, but sent immediately on change |
| odometer | * Numerical, km 0 to 16777214.000 km, with about .2m resolution * 10Hz |
| ignition\_status | * states: off, accessory, run, start * 1Hz, but sent immediately on change |
| fuel\_level | * percentage * 2Hz |
| fuel\_consumed\_since\_restart | * numerical, 0 - 4294967295.0 L (this goes to 0 every time the vehicle restarts, like a trip meter) * 10Hz |
| door\_status | * Value is State: driver, passenger, rear\_left, rear\_right. * Event is boolean: true == ajar * 1Hz, but sent immediately on change |
| headlamp\_status | * boolean, true is on * 1Hz, but sent immediately on change |
| high\_beam\_status | * boolean, true is on * 1Hz, but sent immediately on change |
| windshield\_wiper\_status | * boolean, true is on * 1Hz, but sent immediately on change |
| latitude | * numerical, -89.0 to 89.0 degrees with standard GPS accuracy * 1Hz |
| longitude | * numerical, -179.0 to 179.0 degrees with standard GPS accuracy * 1Hz\ |

## Vehicle Parameters to upload to backend

* Upload all signals
* Upload selective signals
  + Below signals as more appropriate candidate for upload
    - * Latitude
        + numerical, -89.0 to 89.0 degrees with standard GPS accuracy
        + 1Hz
      * Longitude
        + numerical, -179.0 to 179.0 degrees with standard GPS accuracy
        + 1Hz\
      * vehicle\_speed
        + numerical, 0 to 655 km/h (this will be positive even if going in reverse as it's not a velocity, although you can use the gear status to figure out direction)
        + 10Hz
      * Timestamp
      * Vehicle direction – ***possible to be deduced from the lat/long***

## Vehicle co-ordination that selected parameters could enable

Locations address can be derived using Latitude and Longitude and current location of health worker can be identified. By using vehicle\_speed parameters time of arrival of health worker could be estimated, as in rural area vehicle speed depends on conditions of road and traffic.

## Devices perspective in vehicle

### MOTECH Phone

J2ME mobile device is supported in Motech setup. Once the drive traces data is saved to Motech server, the relevant information can be fetched on J2ME phone as well, which are being used by health workers.

### OpenXC device

Using Android mobile device health workers will get registered on Motech server and vehicle drive traces data (selected parameters) will be pushed using health worker’s id, so that health worker’s location could be identified.

## Possible technical Modes of interface between OpenXC device and Backend

These are the possible technical modes of interface between OpenXC device and backend:

1. REST: Rest web service can be used to sync OpenXC database and Motech database, also to sync SQLite database (Android device cache) and OpenXC database.
2. HTTP: Http protocol can be used to display all relevant information about health worker’s location and average vehicle speed etc.

**Data format:** JSON can be used as data format to send and receive data.

## Data caching aspect at backend

Data caching is also possible, if needed. Drive traces data can be stored in SQLite server. It could be needed in case, if it’s required to know the route of health worker’s vehicle and internet connectivity is not stable.

In such cases data could be stored in SQLite server locally on Android device and later on data could be synced with the OpenXC database using web services.

## Possible Apps that could be developed at phone end

1. At J2ME phone: There will be an application for J2ME phone, which will fetch the data from OpenXC database and showing information about health worker, i.e. location of health worker, vehicle speed etc., so that health workers can coordinate with each other.
2. At Android phone: Android phone will be having application to verify the details of health worker and push the drive traces data to OpenXC server. For cache purpose data can be stored on SQLite database and be synced with OpenXC database. And the same application will display the details of health workers.

## Data Push model

Drive traces data will be pushed using android device after registering to Motech server by health worker using his/her own id. Only selected parameters will be pushed, which are helpful to trace the current location and expected time of arrival of health worker. Data will be updated after a particular time interval.

In offline mode drive traces data can be stored in local Database SQLite on Android mobile device. And later on when the internet is available, this data from SQLite database can be synced, using web services.

# References

* <http://www.motechsuite.org/>
* <http://www.motechproject.org/>
* <https://code.google.com/p/motech/>