Project # 643735
www.do-change.eu

Do CHANGE Application Programming Interfaces Design – Phase 1

[Deliverable 4.20 (D71), Revision 1.0]

Key Information from the DoA	
Due Date	30-Nov-2016
Type	Report
Security	Public

Description:

This document presents the design for an API that facilitates the asynchronous distribution of information within the Do CHANGE ecosystem, such that the evolving requirements of patients with respect to privacy and data access can be honoured. This will be a key element of the system employed during the phase 2 clinical field trials.

Lead Editor: Idowu Ayoola (OMNI) **Internal Reviewer:** Robert Smith (DOC)

Versioning and contribution history

Version	Date	Author	Partner	Description
0.1	14-Nov 2016	RS	DOC	Initial document skeleton setup.
0.2	25-Nov 2016	IA	OMNI	Added body of document.
0.3	25-Nov 2016	RS	DOC	Reviewer's updates.
0.4	25-Nov 2016	RS	DOC	Added missing information for Appendix A provided by
1.0	25-Nov 2016	AB	SmH	Official release

Statement of originality:

This deliverable contains original unpublished work except where clearly indicated otherwise. Acknowledgement of previously published material and of the work of others has been made through appropriate citation, quotation or both.

Executive Summary

Referring to the DO CHANGE architecture document (Ref. [A]) this document presents the application programming interface (API) to be employed within the Do CHANGE ecosystem for the sharing / distribution of information. In this context, it defines a preferred API design based on WebSockets (Ref. [F]), which allows information to be published, with changes in its content or the wishes of its owner in terms of privacy and sharing, being efficiently synchronised with those subscribing to the published information. The document also specifies APIs for systems that are unable to support WebSockets and provides best practice guidance on security.

This document taken in combination with other Do CHANGE design documents defines the ecosystem architecture that will be implemented for use in the 2nd phase of Do CHANGE clinical field trials.

Table of Contents

Abbreviations

API	Application Programming Interface
DDP	Distributed Data Protocol
HTTP	Hypertext transport protocol
HTTPS	Secure version of the hypertext transport protocol
JSON	JavaScript Object Notation
REST	Representational State Transfer
SSL	Transport Layer Security
TLS	Secure Socket Layer

References

[A] eSMART deliverable D4.9 (D60) Overall Do CHANGE Ecosystem Infrastructure Architecture

[B] eSMART deliverable D4.14 (D65) Do CHANGE Data Dictionaries and Third Party Interfaces Design

[C] eSMART deliverable D4.10 (D61) The Identity and Permissions Management Framework Design.

[D] DDP overview - https://meteorhacks.com/introduction-to-ddp/

[E] JSON Wikipedia page - https://en.wikipedia.org/wiki/JSON

[F] RFC6455 The WebSocket Protocol - https://tools.ietf.org/html/rfc6455

[G] SockJS project page - https://github.com/sockjs

[H] REST Wikipedia page - https://en.wikipedia.org/wiki/Representational_state_transfer

[I] Web Services Wikipedia page - https://en.wikipedia.org/wiki/Web_service

[J] TLS / SSL Wikipedia page - https://en.wikipedia.org/wiki/Transport_Layer_Security

1. About This Document

Do CHANGE aims to develop and provide a health infrastructure for integrated disease management of citizens with high blood pressure, patients with ischemic heart disease and those with heart failure. In this context, it aims to give them access to a range of personalised health services intended to encourage behaviour change and to allow their behaviour and clinical parameters to be monitored. The information collected will be shared amongst the health services in a manner that reflects the patient's wishes and aims to optimise their health and well-being. This document addresses the application programming interfaces (APIs) to be used to facilitate information sharing within the Do CHANGE ecosystem.

1.1. Deliverable context

Project item	Relationship
Objectives	This document is linked to the following WP4 objectives: To Specify and develop the overall Do CHANGE Ecosystem Architecture, both the services & component infrastructure, the data infostructure and its semantics. • The Data collection, storage, management and communication will take place in a secure patient controlled environment and feedback on transactions will be fed back to the individual. To define the service interfaces of all parties.
Exploitable results	This part of the project is service design. It is anticipated that the services developed from this design would be further developed to commercially exploitable systems.
Work plan	This deliverable is associated with Task 4.5 of WP4.
Milestones	This deliverable is not linked to an overall project milestone but does mark the submission of D4.20 (D71)
Deliverables	This document presents the design for an API that facilitates the asynchronous distribution of information within the Do CHANGE ecosystem, such that the evolving requirements of patients with respect to privacy and data access can be honoured.
Risks	By having this document available, the risk of failure of the technology supporting the Do CHANGE phase 2 clinical trials due to inability to share information between services, should be reduced.

2. Introduction

The Do CHANGE ecosystem (Ref. [A]) is designed as a multi-service platform that can be separated into functional parts. Section 3 of the Do CHANGE project document D4.14 Do CHANGE Data Dictionaries and Third Party Interfaces Design (Ref. [B]), describes the Do CHANGE applications that deal with external interfaces to securely collect user data. In the current document, we describe how the Connect applications works with the Collect service, making it possible to distribute user information to authorized internal services such as to Onmi Analytics, Synergy, patient's portal, etc. In summary, Figure 1 illustrates the context of the Collect service and Connect application.

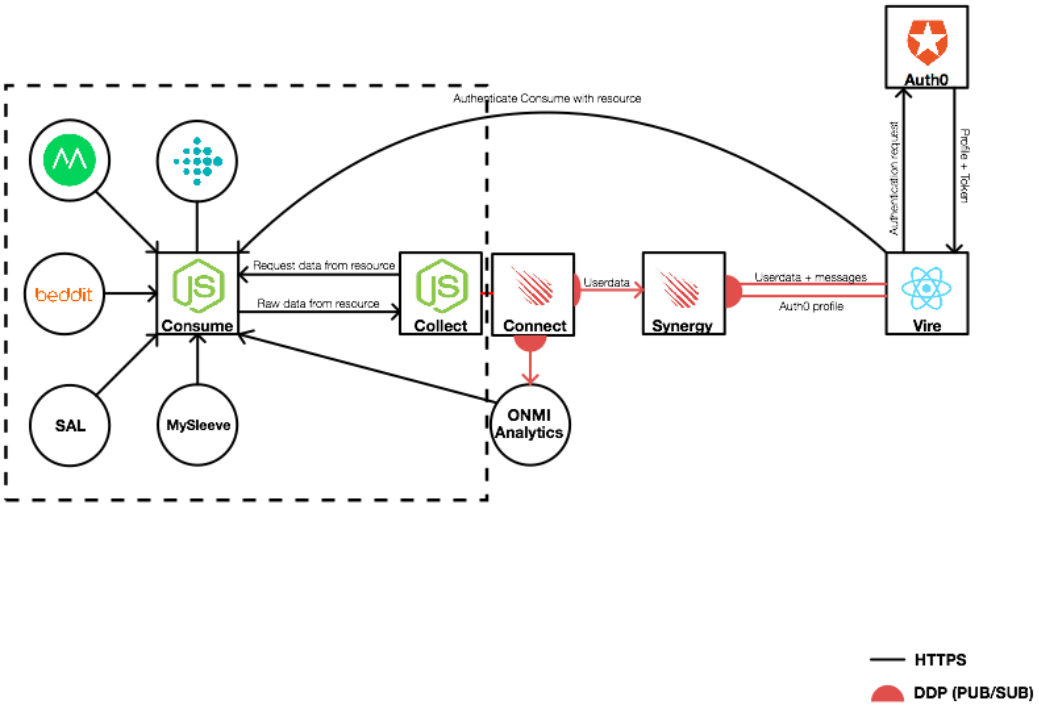

Figure 1 Do CHANGE Connect API context

3. Do CHANGE Distributed Data Approach

The preferred Do CHANGE API design employs a distributed data approach, which follows a publish-subscribe-notification pattern. This pattern makes it possible for data consumers to gain access to patient related information, and for the system to enforce user permissions at all time. The Collect service is the application responsible for publishing user data, which has arrived through the Consume service. Do CHANGE applications may subscribe to the publications. The publications are not implemented by calling a conventional REST (Ref. [H]) or web services (Ref. [I]) based APIs for a list of subscribes. Instead, asynchronous messages are sent to subscribers, with the content of these being dynamically controlled based on the subscriber (the receiver), and its access rights as (Ref. [C]).

3.1. Distributed Data Processes

The processes adopted within the distributed data framework are summarised as follows:

1. The Collect service acting as a proxy "Data Controller" stores information that is to be distributed in a local (private) database and make this information available to other services / applications through its publication mechanism.
2. Consumer applications subscribe to the publications provided by the Collect service.
3. The access rights of the consumer application are checked through the federated resource manager as described in Section 2.3 of D4.10 The Identity and Permissions Management Framework Design (Ref. [C]).
4. A subset of the information subscribed to is sent to the consumer application based in its access right (see step 3).
5. The consumer application maintains a copy of the dataset received in 4 above. Replication involves using specialized software that looks for changes in the distributive database. When changes are identified, the replication process synchronises the copy that each consumer application has.
6. The processing of the replica data set from 4 and 5 above, that is then carried out by the consumer application, must honour the patient's requirements as represented by the Identity and Permissions Management Framework Design (Ref. [C]).

3.2. Technology Adoption

Implementation of the processes described in Section 3.1 above is facilitated using Meteor's Distributed Data Protocol (DDP – Ref. [D]). DDP is the real-time data distribution protocol. It is based on JavaScript Object Notation (JSON – Ref. [E]) and the WebSocket Protocol (Ref. [F] and implemented using the open source SockJS software component (Ref. [G]). DDP is employed to manage the distribution of data-sets through a common API structure. It contains both client and server side elements. The server side makes publications available, while the client can use DDP to subscribe to the publications. The DDP protocol provides three types of notification:

added – this applies when a new item is added to the local data set;

changed – this applies when an item in the local set is updated;

removed – this applies when an item is removed from the local set or publication.

The notifications help to keep the replicated dataset in sync with that from the data provider. Notifications implemented within the client, allow the data consumer application to perform an asynchronous action when the data source changes.

Node.js code snippets are provided in Appendix A for both a server publishing a data-set and a client subscribing to the publication.

3.3. Example

The follow example is provided to illustrate how DDP allows some data in the Behavioural Data Category (see Ref. [A]) to be distributed to Do CHANGE consumer applications.

For this example, we have a data-set called PatientsActivity that holds activity data for all patients. The dataset contains the number of calories burnt, step count and the time of the activity. The Connect application publishes the PatientsActivity data-set and a consumer application, Onmi Analytics, will subscribe to the publication to get activity data and receive data event notifications. Figure 2 illustrates the dataflow, while Figure 3 shows the content of the messages:

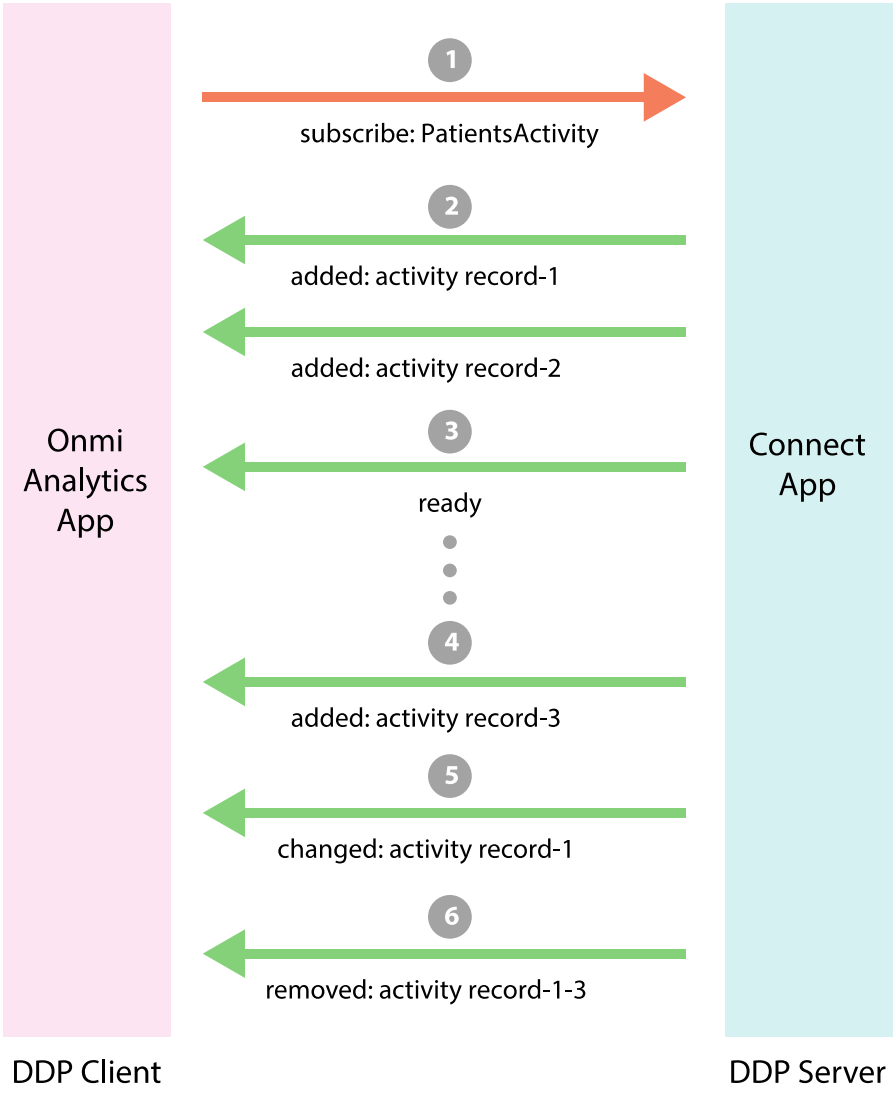

Figure 2 Do CHANGE data sharing based on DDP

```
1.  {"msg": "sub", id: "random-id-2", "name": "activityDataSetPublication",
        "params": [param1, param2]}

2.  {"msg": "added", "collection": "ActivityData", id: "record-1", "fields":
        {"patientId": "ID1", "calories": cal-val1, "stepCount": step-val1, "time": time1}}

    {"msg": "added", "collection": "ActivityData", id: "record-2", "fields":
        {"patientId": "ID2", "calories": cal-val2, "stepCount": step-val2, "time": time2}}

3.  {"msg": "ready": "subs": ["random-id-2"]}

4.  {"msg": "added", "collection": "ActivityData", id: "record-3", "fields":
        {"patientId": "ID1", "calories": cal-val3, "stepCount": step-val3, "time": time3}}

5.  {"msg": "changed", "collection": "ActivityData", id: "record-1", "fields":
        {"stepCount": step-change}}

6.  {"msg": "removed", "collection": "ActivityData",
        id: "record-2"}
```

Figure 3 Example DDP message content

The numbered steps from Figure 2 and Figure 3 are explained as follows.

1. The DDP client (Onmi Analytics) sends the Connect application a subscription request for the data-set PatientsActivity.
2. The Connect App will retrieve all data records that the Onmi Analytics application is permitted to access Behavioural Data for all patients. Hence, the Onmi Analytics application will receive a sequence of “added” notifications containing the existing activity records.
3. After the existing records have been sent by the DDP server (the Connect application), DDP will send a special message called “ready”. The ready message indicates that all the initial data for the subscription has been sent.
4. When new data records are available at the data source, the Onmi Analytics application will receive new records as another “added” notification.
5. The “stepCount” for record-1 changed and the Onmi Analytics application will receive a “changed” notification with the changed field.
6. In the case where the patient with ID2 revokes permission for Onmi Analytics to access their Behavioural Data, the relevant fields are dynamically removed from the publication, and then the Onmi Analytics application will receive a “removed” notification.

All of the above communications takes place asynchronously via a WebSocket connection between the Connect application and the Onmi Analytics application.

4. Use of Conventional REST APIs

While the Distributed Data Approach using DDP described in Section 3 is the preferred API design for information exchange within the Do CHANGE ecosystem, there are some situations where use of this would not be appropriate or possible. Most notably these are:

- When interfacing to existing systems that provide a more conventional API but do not support DDP.
- Where there is not a requirement for asynchronous, real-time communications of information and therefore the resource allocation required to convert a conventional API to one based on DDP cannot be justified.

Where this is the case, more conventional REST APIs are to be employed (Ref. [H]). The functionality provided by these will be specified on a case-by-case basis, but they should all comply with the requirements of the Identity and Permissions Management Framework Design (Ref. [C]).

Note that the need to support web services (Ref. [I]) based APIs is not considered a priority, since their use is in decline with REST APIs being preferred due to their greater efficiency and wide level of support.

5. Security Best Practice

As stated previously, the Do CHANGE applications or services linked to the APIs that are the subject of this document must comply with requirements of the Identity and Permissions Management Framework Design (Ref. [C]). In this context, the following best practices shall be adopted in implementations:

1. Interfaces that allow interactive users to access patient information should implement login authentication with at least minimal strength passwords; any default passwords should be forced to be updated upon first use.
2. Where appropriate, data publication sources should implement IP address white lists to help guard against unauthorised access.
3. Communications interfaces should be based on https or secure sockets, using TLS version 1.1 or later (not SSL see Ref. [J]), employing strong ciphers and using key lengths of at least 256-bits.
4. Personal data “at rest”, that is in persistent storage, should be encrypted using AES-256 encryption or better.

6. Conclusion / Future Work

This document has provided the design for a preferred API to be used for the distribution of data within the Do CHANGE ecosystem, along with guidance on security and dealing with systems that are unable to support the preferred API design. This provides a key element of the overall Do CHANGE ecosystem. The next stage is to implement this design and combine it with other key elements to form a working prototype for the Do CHANGE ecosystem for use in the phase 2 clinical field trials. This working prototype should then be tested and evaluated, making any necessary refinements before commencing the phase 2 clinical field trials.

Appendix A – Reference Code Samples

Overview

In this appendix, sample code for a publisher and subscriber are provided. Both are written in node.js.

Publisher Code Snippet

The following code snippet is an example of publishing user data in Collect.

```
Meteor.publish("userdata.protected", function(authenticationToken){

 /*************************************************************
  * This will watch the entire MongoDB Collection 'UserDatas' *
  *************************************************************/

  if(authenticationToken){

    // Verify the validity of the token received
    jsonwebtoken.verify(authenticationToken, Meteor.settings.private.JWT_KEY, function(error, token) {

      if(error){
        // Token validation failed
        return false //Throw Meteor error
      }

      if(token){
        // JWT is ok
        return UserDatas.find()
      }

    })
  }

  if(!authenticationToken){
    console.log('No authentication')
    return false //Throw Meteor Error
  }
})
```

In the code snippet above, a publication named “userdata.protected” is made. The function receives an authentication token passed by the subscriber to validate the authenticity of the subscriber. The token is a JSON Web Token as described in section 2.3, D4.10 (D61) The Identity and Permissions Management Framework Design (Ref. [C]). This token contains the identity of the subscriber along with claims relating to access and a signature. Hence, this can be used without further references to and identity provider to authorise or otherwise the access.

Subscriber Code Snippets

The code snippets presneted here are an example of subscribing to the user data publication presented in the Publisher Code Snippet above. The code runs on the data consumer application (the client side).

First a local database called ConnectData is created:

```
/*************************************************************
 * New local collection that mirrors the remote collection *
 *************************************************************/

export const ConnectData = new Meteor.Collection('ConnectData')
```

Next, a DDP connection is made with the Do CHANGE Connect application (the server side). The connection object is used to create another database that binds to the DDP connection.

```
/*************************************************************
 * New local collection that mirrors the remote collection *
 *************************************************************/

export const ConnectData = new Meteor.Collection('ConnectData')


/*****************************************************************
      *     DDP.connect() to Connect and assign new remote collection
      *     'userdatas'
  * 'userdatas' is present in remote DB
*****************************************************************/

 const Connect = DDP.connect('https://collect.id.tue.nl/Connect/')
 const ConnectUserData = new Meteor.Collection('userdatas', {connection: Connect})
```

The client application observes changes to the distributed dataset. It will receive notifications when new data is added, changed or removed form the distributed data:

```
 /*****************************************************************
  * Observe remote collection and insert into local collection *
  *****************************************************************/
 ConnectUserData.find().observe({

   added: function(data) {
    // Received newly added data
    ConnectData.insert(data)
   },

   changed: function(data){
    // A record has changed. Update local database
    ConnectData.update(data)
   },

   removed: function(data){
    // A record has been removed. Remove data from local database.
    ConnectData.remove(data)
   }

  })
}
```