

XDN: Cross-Device Framework for Custom Notifications Management

*Original*

XDN: Cross-Device Framework for Custom Notifications Management / Corno, Fulvio; De Russis, Luigi; Montanaro, Teodoro. - In: COMPUTING. - ISSN 0010-485X. - STAMPA. - 101:11(2019), pp. 1735-1761. [10.1007/s00607-018-0686-6]

*Availability:*

This version is available at: 11583/2718416 since: 2019-10-14T15:04:28Z

*Publisher:*

Springer

*Published*

DOI:10.1007/s00607-018-0686-6

*Terms of use:*

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

*Publisher copyright*

Springer postprint/Author's Accepted Manuscript

This version of the article has been accepted for publication, after peer review (when applicable) and is subject to Springer Nature's AM terms of use, but is not the Version of Record and does not reflect post-acceptance improvements, or any corrections. The Version of Record is available online at: <http://dx.doi.org/10.1007/s00607-018-0686-6>

(Article begins on next page)

# XDN: Cross-Device Framework for Custom Notifications Management

Fulvio Corno · Luigi De Russis · Teodoro Montanaro

Received: date / Accepted: date

**Abstract** With the increase of connected devices and online services, and recently IoT devices, the number of notifications received by every person is growing. The importance of notifications, as they become part of people's lives, often depends on various factors that can differently influence the reaction and the disruption of recipients. For this reason, the generation and the distribution of notifications has been gaining importance in the design of new applications, services, and smart devices. Nowadays, developers have not yet fully exploited all the advantages that the availability of multiple devices could bring in the customization and the distribution of notifications, e.g., exploiting a cross-device approach.

This paper presents XDN (Cross Device Notification), a framework to assist developers in creating cross-device notifications by scripting. The XDN architecture includes: a library to enable developers to design personalized notifications to be distributed among ad-hoc networks of IoT/mobile devices; a GUI to assist developers in implementing and testing (through a simulator) notification strategies; a server-side runtime environment; and an XDN IoT/mobile library for device support.

We discuss the requirements for cross device notification systems, and we present the features of the XDN framework, in particular from the point of view of developer advantages, validated through different scenarios.

**Keywords** Cross-Device · Framework · Notifications · Developer · API

## 1 Introduction

During the last decade, the presence of notifications in people's routines has grown: people receive, in fact, a huge and increasing number of notifications generated by different sources (e.g., instant messaging apps, cloud services, Internet of Things devices) at any time with the aim of facilitating their lives [14]. Although people are

---

F. Corno · L. De Russis · T. Montanaro

Politecnico di Torino, Dip. Automatica ed Informatica, Corso Duca degli Abruzzi 24, 10129 - Torino, Italy  
E-mail: {fulvio.corno, luigi.derussis, teodoro.montanaro}@polito.it

becoming accustomed to them, the usefulness and the importance of each notification often depends on various factors that can influence the reaction and the disruption of recipients. The information contained in notifications, or the device(s) on which they are presented, are only a few examples of factors that can influence the user reaction and disruption caused by notifications.

As declared by Seshadri et al. [18], in fact, even though providing individuals with relevant information is an essential element in facilitating their activities, the challenge for developers is “to provide information in a desired manner notwithstanding vast differences in individuals’ information and delivery preferences”. Developers should create applications that “deliver timely and personalized information on whatever suitable device is available and accessible” [18].

Furthermore, the Internet of Things (IoT) is also gaining importance in the notification context. The IoT is the network of physical objects always connected to the Internet with the aim of sharing services and information with other connected “things”. Its growing spread is introducing new devices every day and, as declared by Weber [19], “the ongoing wave of smart devices makes it possible to reach the user through multiple devices at once, amplifying the effects of notifications”. Nonetheless, in recent years, this possibility has not been fully exploited by developers: most existing applications mainly duplicate the same notification on all the available supported devices.

While it is necessary to personalize notifications according to their importance, the developed notification strategies (i.e., algorithms for distributing notifications) should exploit the possibility of reaching the same user through different devices. As already proposed [6, 10], a possible contribution entails the adoption of a cross-device approach [5] to notifications, a growing trend of the last decades that consists of extending an application user experience across multiple devices. By applying the cross-device approach to notifications, in fact, developers could distribute different “signals” related to the same notification on different devices. For instance, a warning sound could be sent to the smart Hi-Fi, while vibration could be activated on the personal smartphone and the notification content could be shown on the smart TV.

Developers should therefore focus on both personalizing notifications to differentiate the presentation of important and unimportant information, and designing cross-device notifications strategies responsible for informing users without causing too much disruption and involving mobile and IoT devices. Nonetheless, as highlighted by related works, developers are not yet supported in implementing solutions that respect both needs in all their aspects.

This paper presents XDN (Cross Device Notification), a framework that allows developers to create, by scripting, cross-device notifications. Inspired by the Chord framework [22]<sup>1</sup> and with the aim of contributing to its future development (if the project will be revived), XDN assists developers in a) designing personalized notifications, and b) designing, implementing and testing notification strategies able to distribute notifications among mobile and IoT devices using a cross-device approach. The framework architecture is composed of four main parts:

<sup>1</sup> <https://github.com/google/chord>, last visited on July 10, 2017, last updated on December 5, 2015

- the XDN library, that implements a set of high-level APIs to let developers a) handle incoming notifications and their properties (i.e., content, receipt date/time, generator, and icon), b) select devices, also through their properties and status to be involved in the notification distribution, and c) separately perform different actions on selected devices (e.g., play a sound, activate the vibration, or show the notification content);
- the XDN runtime environment, that is a service supposed to be run on a server and that is able to a) accept notifications from IoT/mobile devices, b) process them by running the deployed script and c) distribute processed notifications to available devices through its dispatcher module. It is also responsible for d) registering new devices and e) storing device status updates;
- the XDN GUI that provides a) an editor for allowing developers implement and debug their notification strategies and b) a simulator that shows how the pre-defined devices will behave when a new notification arrives by simulating the runtime environment;
- the XDN IoT/mobile library that will be imported in every application/service that uses XDN to generate and distribute notifications. It allows developers to a) generate notifications compatible with the JSON format supported by XDN, and b) send the generated notifications to the XDN runtime environment. In addition, it is also able to autonomously c) receive commands from the XDN runtime environment, and d) execute them.

A first prototype of the XDN framework, that implements only the XDN library and the XDN GUI was developed in the Node.js framework. Consequently, the methods, classes and objects provided by the XDN library are provided for JavaScript applications. Furthermore, the feasibility of the framework and the fulfillment of all the requirements presented in Section 4 section were verified through the simulation of realistic scenarios.

The main contribution of this work is a) a new cross-device proposal for customizing and distributing notifications among ad-hoc networks of end-user mobile and IoT devices; b) the XDN framework, that provides the APIs, the GUI, the runtime environment, and the IoT/mobile library for developers that would develop their algorithms respecting the cross-device approach; c) two different scenarios implemented as realistic notification strategies to verify the feasibility of the framework and the fulfillment of all the requirements.

## 2 Background

With the aim of helping readers in understanding the situations in which the XDN framework would help developers, this section presents a sample notification strategy (i.e., an algorithm developed with the aim of smartly distribute the incoming notifications) that will be also used as a running example through the remainder of the paper.

## 2.1 Scenario: Messaging Application

Three different specific scenarios, inspired by the motivating scenarios presented by Campbell et al. [10] will be used to extract the final general strategy.

Ashley is a developer, working on a messaging application that should differently disrupt users depending on their current activity and location. She identified three sample situations that should be handled by her algorithms, considering some hypothetical users of her application.

- The user is at her business location, is wearing a smartwatch and is using her desktop computer. Unfortunately, her smartphone went out due to low battery and, suddenly, an important personal message arrives. The user should be warned about that message as soon as possible by showing it on the desktop computer that he is using and by making the smartwatch vibrate. In addition, no other persons should be disrupted (e.g., by playing a warning sound for more than one time on other devices).
- The user is at home and her smart TV is playing a movie. The user owns one smartphone. It is supposed that the user is relaxing in front of the smart TV, so the system should not cause any disruption to him. In this situation, a notification should be shown on the smart TV, while the smartphone vibration should be also activated just because the user could not be actually in front of her TV.
- The user is driving. The car is equipped with an IoT Hi-Fi system, i.e., an Hi-Fi that is connected to the Internet to provide different services. One of these services consists of loudly reading incoming notifications. While the user is driving, he receives two notifications on her smartphone: one is from her daughter who is waiting her in front of the school, while the other one is an advertisement. The system should a) generate a textual notification on user smartphone for both received messages, b) activate the vibration on the smartphone to inform the user about the daughter's message, only, and c) read the daughter's message on the car Hi-Fi.

The above situations are not exhaustive, for this reason, in all the situations that are not covered by the described ones, the smartphone will be used as the preferred device: the notification content will be shown on the smartphone display, the vibration will be activated and the smartphone LED will be made blink.

Ashley identified the following notification strategy expressed in term of general rules that she will implement. In the following list the adjective “available” will be used to indicate a device that is turned on and connected to the internet.

- If a message is received from a service that inserted the words “@important” and “@personal” in the notification metadata and the user is working (i.e., no smart Tv and/or Hi-Fi and/or car Hi-Fi is available, while a PC is available), the message should be notified by showing the notification content on all the available PCs and tablets and the vibration should be activated only once and on all the available devices that support it;
- If a message is received when the user is relaxing in her house and in front of her smart TV (i.e., the status of a presence detector installed near the TV reveals the

user presence, or one smart TV or one Hi-Fi system is available and is playing music/video), the notification content should be shown on the available TVs, and the vibration should be activated on all the available devices that support it (just in case she is inattentive);

- If a message is received when the user is driving (i.e., the car Hi-Fi system is available and playing music), the system should distinguish useful and useless messages. It is supposed that the app generating the message inserts the word “@unimportant” in the notification metadata, when needed. If the message is not important, only a textual notification will be shown on the user smartphone. Otherwise, the following actions will be performed: a) a textual notification will be generated on all the available smartphones, b) the vibration will be activated on all the available smartphones, c) the notification content will be read on the car Hi-Fi system.
- In all the cases that are not covered by the previous ones, a) a textual notification will be shown on all the available smartphones, b) the vibration will be activated on all the available smartphones, c) the smartphones LED will be made blink.

In addition, Ashley must be able to develop the just described rules without moving from her workplace (e.g., to test the situation in which the user is driving) and to test them without actually owning the involved devices. Consequently, she needs a tool to: a) design and develop the notification strategies able to manage the described situations, b) simulate the behavior of devices that she does not actually own by simulating the arrival of notifications in all the presented realistic situations.

### 3 Related works

Due to the increasing presence of notifications in people’s lives, user frustration and/or disruption caused by notifications have been examined in the literature [3, 12, 9, 1]. Some works also proposed solutions to mitigate the negative effects that notifications cause on user attention [8, 15, 13, 2, 22, 4]. Almost all these solutions rely on the customization of notification strategies at the distribution level (i.e., notifications are intercepted and then systems decide if, when, and how showing them). Instead, in our opinion, another approach is also feasible: the customization of notification strategies at the design level (i.e., notification strategies are designed with the aim of reducing user disruption).

Specifically, nowadays, developers misuse notifications and generate/show them on almost every available device and in every moment of the day, with no uniform mechanism for considering the importance of the notification and/or the user availability. For this reason, we believe that developers should design their algorithms so that notifications could be distributed according to a well-designed and accurately tested strategy. Different works [15, 22, 20], for example, as a result of their experiments report some issues about user attention and/or preferences that developers could consider in implementing their notification strategies with the aim of reducing the overall user disruption. Likewise, some commercial systems have already improved notification distribution strategies at the design level: Slack<sup>2</sup>, a cloud-based

<sup>2</sup> <https://slack.com/>, last visited on July 10, 2017

Feature	Description
F1	Design and develop notification strategies
F2	Support IoT devices as receipt of notifications
F3	Simulate selected device(s) (not owned)
F4	Simulate more than one device at a time
F5	Multi-device
F6	Multi-platform
F7	GUI for developers
F8	Support cross-device approach

**Table 1** Summary of features provided by related works

team collaboration tool that generates a huge number of notifications a day, for instance, already implemented a notification strategy<sup>3</sup> that “smartly” distributes notifications to a cleverly chosen subset of available devices. However, the complexity of such techniques could cause different problems in managing and testing them without a dedicated tool.

The community of developers needs a framework that allows developers to design, develop and finally test their own algorithms to generate customized notifications and distribute them among available mobile and IoT devices using a cross-device approach.

To the best of our knowledge, no work provides such a similar support, consequently, the analysis of related works is split in two different topics: Section 3.1 treats the development of applications, services, or systems able to generate and/or distribute cross-device notifications among available mobile and IoT devices, while Section 3.2 treats the development of a framework for developers for customizing notifications and/or their distribution.

Both subsections discuss the drawbacks and problems of the related works with respect to the notification strategy presented in Section 2. In addition, Table 2 summarizes all the presented related works with respect to the exposed features and services reported in Table 1.

### 3.1 Frameworks and tools for developers for managing notifications

As already mentioned, notifications have been extensively examined in the literature, but only a few works provide developers with aids for customizing notifications and/or their distribution. The main characteristics that distinguish almost every presented work from XDN are related to the absence of a simulator or the impossibility of supporting cross-device interactions.

An interesting work specifically proposed for designing and deploying notification strategies is presented by Kubitza et al. [11]. They describe an infrastructure for homes and offices that allows designers and web developers to design and deploy context sensitive notification strategies using arbitrary things and smart home

<sup>3</sup> <https://slack.engineering/reducing-slacks-memory-footprint-4480fec7e8eb#8c3c>, last visited on July 10, 2017

Related work	F1	F2	F3	F4	F5	F6	F7	F8
meSchHub [11]	Yes	Yes	No	No	Yes	Yes	No	No
Seshadri et al. solution [18]	Yes	No	No	No	No	No	Yes	No
Apple Framework	Yes	Partially	Yes	No	No	No	Yes	No
Google Framework	Yes	Partially	Yes	No	No	No	Yes	No
Apache Cordova Framework	Yes	Partially	Yes	No	No	Yes	Yes	No
AllJoyn Project	No	Yes	No	No	Yes	Yes	No	No
Panelrama [21]	No	Partially	No	Yes	Yes	Yes	No	Yes
XDStudio [16]	No	Partially	Yes	Yes	Yes	Yes	Yes	Yes
Connichiwa [17]	No	Partially	Yes	Yes	Yes	Yes	No	Yes
Notification Platform [6]	No	No	No	No	Yes	Yes	No	No
Campbell et al. solution [10]	No	No	No	No	Yes	Yes	No	No
Chord [22]	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes
XD-Testing [7]	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes

**Table 2** Summary of related works' features

products connected to their meSchHub gateway, such as TVs, tablets, projections, lamps, speakers and many more. In the proposed infrastructure, the notifications received on the user smartphone are sent to the meSchHub gateway that forwards them to available IoT devices according to interaction scripts pre-defined by developers. One gateway can be set up per each smart space (e.g., office, flat, house) and, in the proposed architecture only a single smartphone can be connected to a single gateway. When a smartphone leaves the range of a certain smart space (e.g., out of home WiFi) and comes into the range of another known smart space (e.g., office WiFi) the app automatically discovers the local meSchHub gateway and starts working with the interaction scripts that are defined for that space. According to the description of the system, the meSchHub system lacks a simulator: Ashley, the developer of the “Messaging Application” scenario, in fact, would not be allowed to test her notification strategies on devices that she does not actually own. In addition, the meSchHub system is limited to the environment in which the gateway is installed and in which the user is currently present.

Another interesting related work is the patent proposed by Seshadri et al. [18] that presents a system and a methodology to facilitate the development, debug, and deployment of a notification platform application. The whole system is based on an Application Definition File (ADF) which describes all the components that interact to perform notification services, wherein the components are often in various languages and formats. In their proposal, a visual user interface is provided to facilitate efficient design, debug, management and deployment of an ADF and related configuration file (and other related files) when developing notification applications. Developers are, in fact, directed through visual diagrams and processes leading to the development and ultimately deployment of a notification application. However, the proposed system does not provide any simulator to test the designed algorithms. In addition, according to the description of the system, it does not support any cross-device interaction (e.g., activate the vibration on a device and make the LED blink on another device at the same time).



Moreover, in addition to the solutions found in the literature, developers are supported by all the existing commercial frameworks and/or APIs used in the development of mobile and/or IoT applications. The following analysis will focus on the three most used frameworks/tools for mobile devices development, but the reported observations also apply to other existing solutions that provide the same or similar features: a) the frameworks proposed by Apple for managing notifications within iOS, tvOS, watchOS or macOS applications, b) the framework proposed by Google for the customization and distribution of notifications within Android applications, c) the framework proposed by Apache Cordova (formerly PhoneGap) for the distribution of notifications across multiple platforms (i.e., Amazon Fire OS, Android, BlackBerry 10, Browser, Firefox OS, iOS, Tizen, Windows Phone 7 and 8, Windows). All of them provide support for managing a single device at a time, thus it is not possible to develop cross-device strategies nor to simulate more than one device at a time.

Apple provides two different frameworks for customizing notification in the development of mobile applications: the User Notifications framework that mainly handles the content of the notifications, and the User Notifications UI framework, available only in the iOS SDK (i.e., currently it is not usable for the development of tvOS, watchOS, and macOS apps), that mainly handles the appearance of the notifications. Even though, in some cases, the notification appearance can be customized, developers can only choose how the user should be notified by selecting one of the following options for delivering the notification: a) an onscreen alert or banner, b) a badge on the app's icon, c) a sound that accompanies an alert, banner, or badge. On the other hand, Google provides a similar solution for Android devices but allows more customization<sup>4</sup>. In the development of Android applications, in fact, it is possible to personalize: a) the notification content, b) the notification icon, and c) the notification priority. In addition, in the upcoming "O" release of Android, the following customizations will be added: a) specify the notification channel at which the notification belongs, b) remove or update a snoozed notification, c) set a timeout for creating a notification after a specified time, d) set and enable a background color for a notification, e) add some style to the notification, and f) know the user reaction to a notification (i.e., dismissed or not).

Furthermore, Apache Cordova (formerly PhoneGap)<sup>5</sup>, one of the most used frameworks for building cross-platform applications, provides the "cordova-plugin-dialogs" plugin<sup>6</sup> for customizing notifications. The supported methods mainly provide the following customizations: a) specify the message of the notification, b) specify the title of the notification, c) specify the function that should be invoked when the user presses on the notification, d) specify the function that should be invoked when the user presses on a button present within the notification. In addition, it is possible to play a sound when the notification arrives on all the supported platforms except Firefox OS and Windows (it is possible only on Windows 8).

<sup>4</sup> <https://developer.android.com/guide/topics/ui/notifiers/notifications.html>, last visited on March 22, 2017

<sup>5</sup> <https://cordova.apache.org/>, last visited on May 02, 2017

<sup>6</sup> <https://cordova.apache.org/docs/en/latest/reference/cordova-plugin-dialogs/>, last visited on May 02, 2017

The provided description of such solutions highlights that it is already possible to personalize notifications, even though some limitations can be revealed with respect to the customization we are proposing (e.g., it is not possible to activate only the vibration). However, the main disadvantage of such solutions is that it is not possible to develop notification strategies implementing the cross-device approach. Likewise, some other drawbacks emerge from this brief analysis. First of all, the first two presented commercial frameworks are strictly limited to the development of applications for specific devices and/or platforms. Thus, if a notification strategy is developed for a platform (e.g., Android) it is not easy, then, to export and use it in other platforms (e.g., iOS). In addition, the available developer tools (e.g., Android Studio<sup>7</sup> for the development of Android applications, XCode<sup>8</sup> for the development of iOS, tvOS, watchOS and macOS applications) and the PhoneGap Developer App does not allow the simultaneous simulation of different devices to test the designed notification strategy on all target devices.

Finally, another interesting project is the one proposed by the AllSeen Alliance: AllJoyn<sup>TM9</sup>. Even though, at the moment of writing, the documentation<sup>10</sup> is under review<sup>11</sup>, the AllJoyn<sup>TM</sup> Project seems to fully support notifications by providing a common way for devices to send and receive information directed to users. However, although multiple devices can be used as receipt of the notifications, the messages are broadcast to all the devices connected on the AllJoyn<sup>TM</sup> network without any possibility of customizing the notification strategy. In addition, it does not provide any graphical tool to design and test notification strategies and does not allow the simultaneous simulation of different devices to test the designed notification strategy on all target devices<sup>12</sup>.

### 3.2 Cross-device interactions

A growing trend in ICT is the use of the cross-device approach in the development of applications [5]. It consists of extending an application user experience across multiple devices. The development of cross-device applications has already been applied in different domains to solve different problems, but only a limited number of works are devoted to the generation and/or the cross-device distribution of notifications among different devices.

<sup>7</sup> <https://developer.android.com/studio/index.html>, last visited on March 22, 2017

<sup>8</sup> <https://developer.apple.com/xcode/>, last visited on March 22, 2017

<sup>9</sup> <https://openconnectivity.org/developer/reference-implementation/alljoyn>, last visited on May 03, 2018

<sup>10</sup> <https://github.com/alljoyn/extras-webdocs/blob/master/docs/learn/base-services/notification/index.md>, last visited on May 03, 2018

<sup>11</sup> Due to the merge of the AllSeen Alliance, who sponsored the AllJoyn Project, and the Open Connectivity Foundation (OCF), all the AllJoyn documentation was moved to new locations and some links were not yet updated after the migration

<sup>12</sup> The reported description of the AllJoyn framework may change in the next future: the website reports the intention of a future migration of AllJoyn to the OCF specification that does not provide any support for sending notifications to users

Three of the most popular frameworks that facilitate the creation or the conversion of cross-device applications do not, in fact, support notifications at all. One of them is presented by Yang et al. [21]: Panelrama. It is a framework that facilitates the creation and the easy conversion of existing web applications to enable cross-device interaction. In brief, an application is decomposed in a set of panels, that are distributed among all available devices and properties and statuses are synchronized. The main advantage of Panelrama is that applications can be tested, and also deployed on every available device that is equipped with a browser. However, a) it is not possible to test the designed application on not owned devices, b) until the moment of writing, neither the simulation of IoT devices nor the management of IoT-dedicated properties and/or hardware accessories (e.g., the vibrator) are supported by existing browsers and, consequently, by Panelrama. Moreover, as already mentioned, according to the provided description, Panelrama is not designed to manage or customize notifications. A possible solution to the absence of a support for notifications could resort to the use of Web Push Notifications inside the browsers by using standard Web Notifications<sup>13</sup>, proprietary APIs (like Mozilla Notification API<sup>14</sup>) or public libraries (like Roost<sup>15</sup> or Push.js<sup>16</sup>), but, in this way the notifications could not be received when the browser is closed.

The second and the third frameworks able to facilitate the development of cross-device applications are XDStudio [16] and Connichiwa [17], but, unfortunately, they lack support for notifications. XDStudio provides a GUI builder designed to support interactive development of cross-device web interfaces. The most important advantage with respect to Panelrama is the presence of a simulation tool that allows developers to design algorithms for a multi-device environment and test them on devices that are not owned by the developer. Connichiwa [17], instead, is a versatile framework for creating web applications across multiple devices. It is based on an event-based mechanism to imperatively show and hide content on devices upon connection or disconnection of other devices [7]. Both of them are designed to mainly support the development of user interfaces and do not handle notifications, too.

Consequently, the considerations reported for Panelrama could be extended for XDStudio and Connichiwa, too: a) notifications could not be used when the browser is closed, and b) XDStudio and Connichiwa are not able to manage IoT-dedicated properties and/or hardware accessories (e.g., a LED or a vibrator).

Two works that are specifically designed for notifications are described in the following. Although the authors declare that their proposals are designed to support cross-device notifications, the description does not provide any evidence of it. In fact, to the best of our knowledge, both works provide only support for multi-device notifications, but do not support developers in distributing different “signals” related to the same notification on different devices.

First, Horvitz et al. [6] present the Notification Platform, a cross-device messaging system that modulates the flow of messages from multiple sources to other de-

<sup>13</sup> <https://www.w3.org/TR/notifications/>, last visited on March 22, 2017

<sup>14</sup> <https://developer.mozilla.org/en-US/docs/Web/API/notification>, last visited on March 22, 2017

<sup>15</sup> <https://goroost.com/>, last visited on March 22, 2017

<sup>16</sup> <https://nickersoft.github.io/push.js/>, last visited on March 22, 2017

vices by performing ongoing decision analysis. Specifically, it balances the costs of disruption with the value of information from multiple message sources. The system employs a probabilistic model of attention and executes ongoing decision analyses about ideal alerting, fidelity, and routing. The main drawback of this work is that the developer would not be allowed to modify in any way the predicted model. In fact, it is not possible to customize the notification distribution strategies. In addition, according to the provided description, they apply the cross-device approach to mobile devices only, without considering IoT devices.

Second, Campbell et al. [10] present some techniques for cross-device notifications. They start from the consideration that a notification could be missed for many reasons (e.g., because the smartphone is in a bag) and, even though other devices are in use, the user remains unaware about it. Consequently, they propose a solution that involves available devices to allow user to be warned about arriving notifications. They provide a solution to handle the receipt of a notification from a device different from the one that received it but does not allow to customize the notification or its distribution. Consequently, with respect to the “Messaging Application” scenario, in which Ashley would like to personally design the strategy for distributing the notification and/or its properties, this solution lack of the possibility to do both actions. In addition the proposed technique does not support IoT devices.

More interestingly, Chi et al. [22] present Chord (previously known as “Weave”), a framework for developers to create cross-device wearable interaction by scripting. According to the supplied description, it provides a set of high-level APIs, based on JavaScript, for developers to easily distribute UI output and combine sensing events and user input across mobile and wearable devices. It also contributes an integrated authoring environment for developers to program and test cross-device behaviors, and when ready, deploy these behaviors to its runtime environment on users’ ad-hoc network of mobile devices. The main characteristic of Chord is that it is designed to assist the implementation of cross-device interactions. Thus, it does not treat at all notifications and/or features that are essential to adequately reduce user disruption caused by them. First of all, in fact, in Chord only two characteristics are provided as output of the available devices: the display and the speaker (with a corresponding show and play method), while some other features like vibration and light and the corresponding methods (i.e., vibrate, on, off, blink, ... etc) are not provided. In addition, it is not possible to manage notification properties like the notification content, the notification arrival date, the notification generator.

Finally, Husmann et al. [7] present XD-Testing, a library for testing web-based cross-device applications quite similar to Chord. In facts, it provides a similar data structure and similar related methods, but, in addition, provides a mechanism for specifying formal tests and automating their execution. Furthermore, XD-Testing introduces concepts for implicitly and explicitly selecting devices that are needed, respectively to address specific devices or to dynamically choose the appropriate devices for each command that is executed on them. The absence of an explicit support for notifications is the main drawback of XD-Testing.

In this work we decided to develop XDN to be a) inspired by Chord, mainly due to its cross-device nature, its ease of use and its linear data structure, b) compatible with Chord to be integrated in its future extensions.

## 4 Requirements

This section presents the design-time requirements devised for the XDN framework: they were designed by analyzing both the shortcomings and the successes of existing solutions discussed in the *Related works* section.

### *(R1) API for selecting available devices and executing specific actions*

The two most complete cross-device solutions found in literature are the Chord framework and the XD-Testing library. Even though they do not provide support to manage notifications, they introduce some useful methods and functions that allow developers to easily select available devices and then execute specific actions (e.g., activate the screen). This model was really appreciated by developers that tested their solutions, consequently it will be used as inspiration for providing similar methods and classes in XDN. Specifically, XDN will provide APIs to:

- select available mobile/IoT devices based on their specific properties and status;
- perform specific actions on selected devices (e.g., turn on a LED or make it blink);
- manage notification properties (e.g., arrival time, notification content).

As already done in Chord [22], XDN will be implemented following the “object-oriented event-driven paradigm”. Specifically, inspired by Chord and popular JavaScript libraries such as jQuery<sup>17</sup>, XDN will provide a high-level abstraction for programmers to manipulate notifications and device’s properties and actions.

### *(R2) GUI*

One of the most important features provided within existing works is a Graphic User Interface (GUI) that support developers in a) implementing their algorithms/code, b) visualize the current status of available devices, c) visualize/simulate the behavior of the designed algorithms/strategies. Its presence in almost all the discussed related works, in fact, confirms that it is essential in development tools like the one we are going to develop. Therefore, XDN should provide a GUI to a) develop notification strategies and, then, b) monitor the behavior of the simulated devices performed when one or more new notification is delivered to users.

### *(R3) Notification strategies simulation on not owned devices*

A limitation of some existing works (e.g., [11]) able to simultaneously distribute notifications across multiple devices is the absence of a simulator for testing notification strategies on not-owned devices. Considering that the number of IoT and mobile devices is growing day by day, in fact, developers could not be asked to own every existing device to test their algorithms on them. Therefore, XDN should support the simulation of developed notification strategies on devices that are not actually owned by developers.

### *(R4) Simulation of the designed notification strategies*

The example reported in the *Related works* section, i.e., the notification strategy developed by the Slack platform (that is able to distribute notifications across multiple devices) demonstrates that the complexity of notification strategies is growing. In addition, by introducing the cross-device approach, they will become more complex in the next years and a graphic simulator able to simultaneously show the behavior of more than one device could be really appreciated by developers.

<sup>17</sup> <https://jquery.com/>, last visited on January 15, 2017

Unfortunately, the existing solutions that allow developers to define notification strategies (i.e., the work of Seshadri et al. [18], or commercial solutions like Android Studio or Apple XCode) do not provide any graphic simulator for visualizing such behaviors. Consequently, XDN should provide a simulator able to show the behavior of more than one device performed when a new notification arrives.

*(R5) Multi-platform*

Although different existing works propose some solutions to let developers design cross-device applications and algorithms, they have two main drawbacks related to the distribution of such applications/algorithms. Specifically, at first, most of the discussed solutions propose web-based applications able to distribute notifications within a browser. However, this solution has a main shortcoming: users cannot be reached when the device is not in use and when a browser is not running. Second, the solutions that support the distribution of notifications outside the browser (e.g., development of Android app through Android Studio) mainly allow the creation of software that is strictly limited to some specific devices (e.g., Android devices). Instead, XDN should allow the development of notification strategies that could be easily exported on different platforms and run without using additional tools like a browser.

*(R6) Support for IoT devices*

Nowadays, with the increasing spread of the IoT, new smart devices and appliances are developed everyday with the ability not only to generate but also to show notifications. Consequently, in addition to existing mobile devices, the XDN framework should support existing IoT devices to let developers test their notification strategies.

*(R7) JavaScript*

As already discussed, the two most complete cross-device existing solutions found in literature are the Chord framework and the XD-Testing library. Due to its ease of use and its linear data structure, and considering that we are going to use the same design pattern (as declared in R1), XDN will be designed to be compatible with Chord. Consequently, XDN will provide JavaScript methods and classes to let developers implement their strategies using the JavaScript programming language and let them reuse the implemented code in future version of Chord.

## 5 Framework

The architecture of the XDN framework was designed to satisfy all the reported requirements. XDN was designed to be compatible with the architecture of the Chord framework [22], therefore the syntax of the Chord APIs has guided the design of the XDN APIs.

As shown in Figure 1, the architecture of the framework is composed of four main blocks: the **XDN library**, the **XDN GUI**, the **XDN runtime environment** and the **XDN IoT/mobile library**.

The **XDN library** implements a set of high-level APIs to perform the following actions: a) handle incoming notifications and their properties (i.e., content, receipt date/time, generator, and icon), b) select devices, also through their properties and

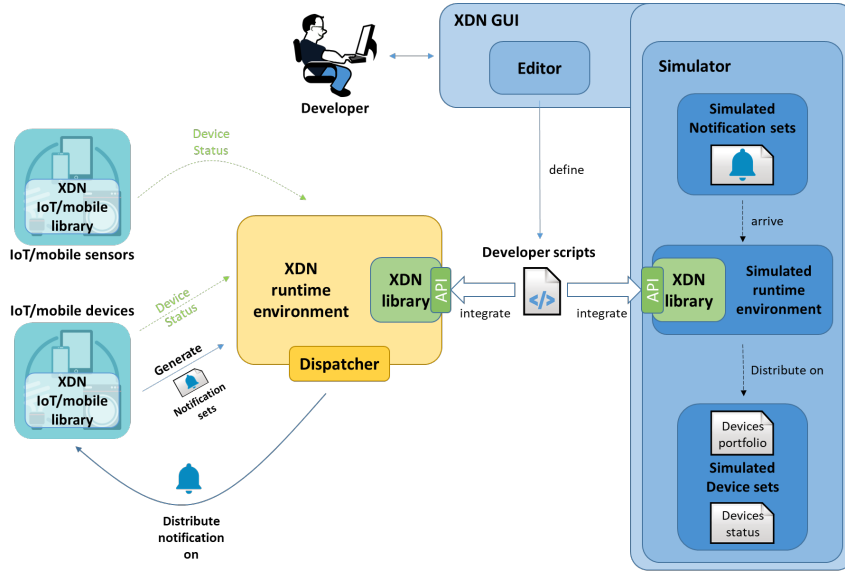


Fig. 1 Architecture of the XDN framework

status, to be involved in the notification distribution, and c) separately perform different actions on selected devices (e.g., play a sound, activate the vibration, or show the notification content). It is mainly designed to be integrated in the XDN runtime environment module and its simulator. The details of the XDN library will be discussed in the following *XDN library* sub-section.

The **XDN runtime environment** is a service that is supposed to be run on a server and to be always available (i.e., turned on and connected). In fact, it is able to a) accept real notifications, b) process them by running the deployed script and c) distribute processed notifications to available devices through its internal dispatcher module. In addition, it is responsible for d) registering new devices and e) storing device status updates. The details of the XDN runtime environment will be discussed in the following *XDN runtime environment* sub-section.

Moreover, developers will mainly interact with the **XDN GUI** module. It provides a) an editor to implement and test notification strategies and b) a simulator to test them. Specifically, the editor provides support to write JavaScript notification strategies or load them from an existing developer script. Meanwhile, the simulator is able to simulate the arrival of notifications by permitting to:

- define or load a device set (better explained in the following description) to be used during the simulations;
- define or load a list of notifications used during the simulation as arriving notifications;
- run the simulation, to actually simulate the arrival of notifications and visualize the behavior (only related to the arrival of a notification) of all the loaded devices.

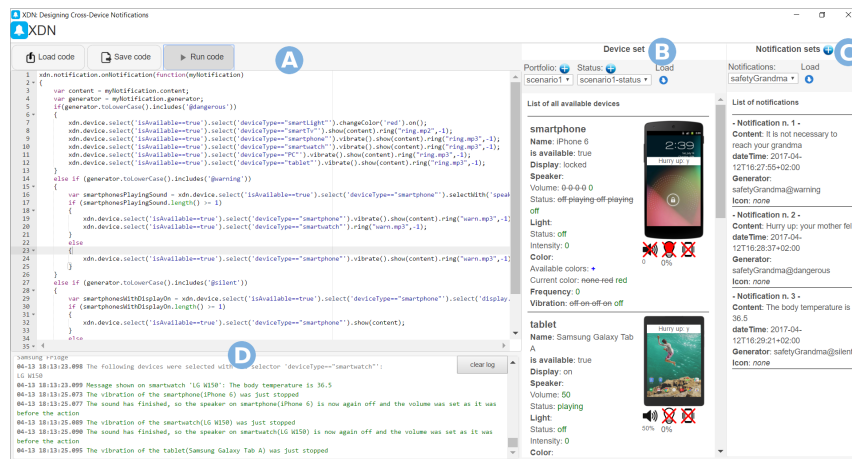


Fig. 2 Screenshot of the XDN GUI

As can be observed in the architecture shown in Figure 1, the simulator relies on *simulated notification sets* and *simulated device sets*. The *simulated notification sets* block represents the list of notifications that will be “sent” during the simulation. While, the *simulated device sets* block represents the group of devices that will be simulated: their behavior corresponding to the arrival of a new notification will be shown during the simulation. A single simulated device set is composed of a) some devices’ properties that are static and represent the device capabilities, and b) the corresponding current devices’ statuses, that represent the current values for each device’s property. The details of the **XDN GUI** module will be discussed in the following *XDN GUI* sub-section.

Finally, the **XDN IoT/mobile library** is the module that is supposed to be imported in every application/service that uses XDN to both a) generate and distribute notifications, and b) update its current status. It allows developers to a) generate notifications compatible with the JSON format supported by XDN, b) send the generated notifications to the XDN runtime environment. In addition, it is also able to autonomously c) update the device current status, d) receive commands from the XDN runtime environment, and e) execute them.

The details of the presented modules will be discussed in the following sub-sections, where the “Messaging Application” scenario (reported in Section 2.1) will be used as a running example.

## 5.1 XDN GUI

Figure 2 shows a screenshot of the XDN GUI. It is composed of four different main parts:



- the script editor (letter A), that allows developers to develop their notification strategies in JavaScript. It integrates graphic warning signals to inform the developer of any syntax error in the implemented code;
- the device set column (letter B), that shows the current status of each loaded device and, if the device supports notifications, simulate its behavior during the simulation of a notification arrival. It also allows to save and load sets of devices to use during the simulation;
- the notification set column (letter C), that shows the list of notifications that will be “sent” during the simulation. It also provides a buttons to save and load a list of notifications;
- the log (letter D), that shows developers any error and warning generated during the simulation and, also, shows the list of all the actions performed on the available devices. Thus, it actually acts as a storyboard of the behaviors of the devices loaded in the device set.

To better clarify the role of each presented component, the “Messaging Application” will be used as a running example to explain the actions that a developer should perform to use the XDN GUI to design, implement and test her notification strategies.

As already presented, in the “Messaging Application” example, Ashley is developing a messaging application that should differently notify users depending on their current activity and location (acquired by analyzing the available devices and their statuses). As a first step, Ashley connects to the web-based XDN GUI and starts from the selection of the devices a user will supposedly own and use. She can choose between two options: select one of the existing set of devices provided in the XDN GUI or create a new custom set of devices. She decides to choose the first option and she selects and loads the predefined set of devices (i.e., a smartphone, a smartwatch, a smart TV, a PC, and a car Hi-Fi). Loaded devices appear in column “Device Set” (B).

Now it is time to write the code she will load on the runtime environment. She can write it by using the editor present in the GUI, located in the left column (A).

The details of the algorithm will be discussed in the following Section 5.2.

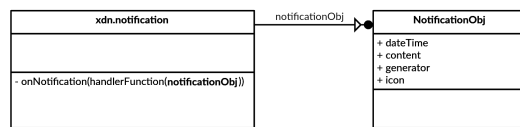
After creating the algorithm, Ashley tests it using the simulator. To do it she has to perform three actions: a) load the initial status of the available devices by using the buttons present at the top of column B, b) load the notifications set that will be sent during the simulation, by using the buttons present at the top of column C and, c) run the simulation using the *Run code* button located at the top of the editor (A).

When all these actions are performed, Ashley will see all the updated statuses in column B and will be able to analyze the list of all the performed actions in the log section (D).

## 5.2 XDN library

The XDN library provides a set of APIs that clearly define methods to customize and distribute cross-device notifications by reducing repetitive code. They are based on two main objects: **x<sub>dn</sub>.notification** and **x<sub>dn</sub>.device**.

The **x<sub>dn</sub>.notification** object is responsible for all the interactions with the notifications and their properties. The class diagram reported in Figure 3 shows its



**Fig. 3** Class diagram that shows the structure of the `xdn.notification` Object

Property	Type
<b>dateTime</b>	date and time at which the notification was received
<b>content</b>	the content of the notification
<b>generator</b>	meta-data associated with the notification, inserted by the generating app
<b>icon</b>	the icon associated to the notification, if available

**Table 3** Notification properties

structure. As shown in the diagram, the **xdn.notification** object provides only the *onNotification* function. It allows developers to attach an event handler to the arrival of a notification. In fact, whenever a new notification arrives, the handler function is called. Thus, developers can implement their notification strategies inside a function that should be, then, passed as input parameter of the *onNotification* method. Notification properties can be accessed by the *NotificationObject* object accepted as input parameter of the handler function. It represents a single notification and contains the properties reported in Table 3. As an example, the code for logging the content of the received notification is:

```

xdn.notification.onNotification(function(myNotification) {
  var content = myNotification.content;
  xdn.log(content);
})
  
```

The **xdn.device** object implements all the classes, sub-objects and methods needed by developers to interact with available devices. The class diagram reported in Figure 4 shows its structure. It is only an abstract object that exposes the methods implemented by the *DeviceSelection* sub-object. The provided methods can be used to a) search and filter across a set of devices, and b) perform actions on one or more selected devices. The list of all the registered devices is stored inside the private *devices* object with their properties, sub-properties and statuses. Each property is assigned to each device depending on its nature: every time a new device is registered to the system (through the XDN runtime environment), only the properties that are supported are specified. As an example, if Ashley is using a smartphone, it has for sure a display and a speaker, consequently the properties *display* and *speaker* will be available.

All possible properties and sub-properties of a device are listed in Table 4 in the *Property* and *Sub-Property* columns. The lower part of the table lists the status properties and sub-properties.

Before interacting with devices it is necessary to select them by using one of the methods listed below. According to the specified criteria they return a *DeviceSelection* object containing the selected *devices*.

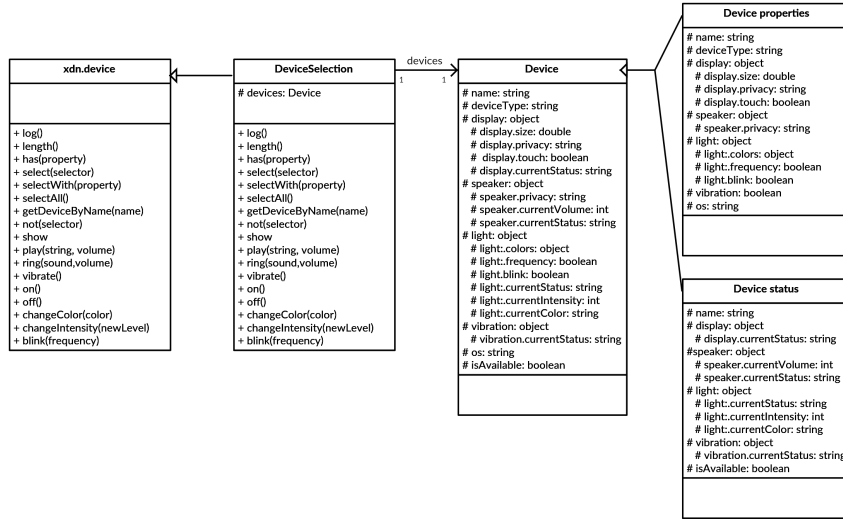


Fig. 4 Class diagram that shows the structure of the xdn.device Object

- *xdn.device.select*: selects the devices that satisfy the specified criteria (e.g., the code `xdn.device.select('deviceType=="smartwatch"')` returns a list of all the smartwatches, that are all the devices with the “deviceType” property set to “smartwatch”);
- *xdn.device.selectWith*: selects the devices that has the specified property (the property is set as input parameter);
- *xdn.device.selectAll*: selects all the devices registered to the system;
- *xdn.device.getDeviceByName*: selects a single device, the one with the specified name (*deviceName* property set to the specified name)
- *xdn.device.not*: selects the all the devices that do not satisfy the specified criteria. This method is used to exclude one or more devices, e.g., if Ashley wants to exclude all the smartwatches: `xdn.device.not('deviceType == "smartwatch"')`.

These methods can be concatenated with a “fluent” programming pattern, so that only the *Device* objects satisfying all the specified criteria are selected. Thus, if Ashley wants to select all the available smartphones, she can use:

```
xdn.device.select('deviceType=="smartphone"').select('isAvailable==true')
```

Furthermore, if she wants to understand if a person is present in a room she can select all the Presence devices that has “true” as “currentStatus” and then count them; if the number is more than 0 then a person is present. These behavior can be performed through the following Listing 1.

Table 5 summarizes all the actions that it is possible to perform on each selected device. It is important to note that the available methods can be used only if the

Property	SubProperty/[value]	Note
<b>name</b>	-	unique identifier
<b>deviceType</b>	[smartphone, smartwatch, bracelet, smartLight, tablet, PC, fridge, hi-fi, smartTv, carHi-fi, smartToothbrush, presenceDetector]	-
<b>display</b>	size, privacy: [high, normal, low], touch: [true, false],	privacy = indicate if the message could be read from only the recipient or also by others
<b>speaker</b>	privacy: [high, normal, low]	privacy = indicate if the message could be read from only the recipient or also by others
<b>light</b>	colors, intensity: [true, false], frequency: [true, false], blink: [true, false]	it indicates both the light of a bulb or the light of a LED (e.g., the status LED of a smartphone)
<b>vibration</b>	[true, false]	-
<b>os</b>	-	-
<b>presence</b>	-	-
<b>isAvailable</b>	[true, false]	- it is set to true if the device is turned on and connected
<b>display</b>	currentStatus: [on, locked, off]	-
<b>speaker</b>	currentVolume, currentStatus: [playing, off]	-
<b>light</b>	currentStatus: [on, off, blinking], currentIntensity, currentColor	-
<b>vibration</b>	currentStatus	-
<b>presence</b>	currentStatus: [true, false]	-

**Table 4** Device properties (above) and statuses (below)

```

var num = xdn.device.select('deviceType=="presenceDetector"').select('
    presence.currentStatus==true');
if (num > 0) {
    // user is present
}

```

Listing 1: Code for acquiring information about user presence in a place

Enabling property	Action
<b>display</b>	.show
<b>speaker</b>	.play, .ring
<b>light</b>	.on, .off, .changeColor, .changeIntensity, .blink
<b>vibration</b>	.vibrate

**Table 5** Device actions

corresponding property (reported in the column “Enabling property”) is defined for the specific device. For example, it is possible to turn the light on with the action *.on* only if a light property is specified for the device. Otherwise, a warning message is be shown in the log section.

Finally, Listing 2 shows the pseudocode of the final complete algorithm written by Ashley to implement the designed behavior of her “Messaging Application” system.

```

Every time a notification arrives
  Get the content and the generator of the notification
  if the generator contains the string '@important' AND the string '
    @personal'
    if user is working ((no smart Tv) and (no Hi-Fi) and (no car Hi-Fi
      ) and (PC) is available)
      Show the content on every available PC and tablets
      Activate vibration on every available device with vibration
    else
      if the user is driving (car Hi-Fi is available and playing music
        )
        Show the content on all the available smartphones
        Activate the vibration on all the available smartphones
        Loudly read the content on the car Hi-Fi
      else
        if the user is driving (car Hi-Fi is available and playing music)
          Show the content on all the available smartphones
        else
          if the user is relaxing at home (presenceDetector reveals user
            presence and (one smart TV or one Hi-Fi system is available
              and playing music/video)
            Show the content on all the available Tvs
            Activate vibration on all the available device with vibration
          else
            Show the content on all the available smartphones
            Activate vibration on all the available smartphones
            Make the LED of all the available smartphones blink

```

Listing 2: Pseudocode for the “Messaging Application” system

### 5.3 XDN runtime environment

The **XDN runtime environment** is a service that is run on a server. Each developer can instantiate a new XDN runtime environment for every notification strategy in her applications. By interacting with the XDN library, it is able to:

- accept registration requests from a device. Every time a new device should be registered to the system, it has to contact the runtime environment providing information about its properties and statuses (Table 4);
- accept update requests arriving from registered devices to inform the system about the changes in their statuses;
- accept new notifications generated by the registered devices. The notification will be accepted in the JSON format, with the fields listed in Table 3;
- customize and dispatch the notifications according to the notification strategy defined by the developer.

## 5.4 XDN IoT/mobile library

The **XDN IoT/mobile library** is the module that developers should integrate in their applications and services for IoT/mobile device to:

- generate notifications as presented in the previous sub-section;
- send the generated notifications to the XDN runtime environment;
- receive commands from the XDN runtime environment, in JSON format containing the properties of Table 5;
- execute the received commands.

Considering that this module should be integrated in almost every existing IoT/mobile application, it is supposed to be developed in different programming languages and for different platforms.

## 6 Implementation and evaluation

To demonstrate the feasibility of the proposed XDN framework and the fulfillment of all the requirements (Section 4), a prototypical version of the XDN framework was implemented, and two different scenarios were designed and implemented.

The goal of the experiments was to collect information, from the developer point of view, about how much effort, in terms of lines of code and spent time, is required by developers to implement their notification strategies. For this purpose, the XDN prototype, currently integrates the XDN GUI and the XDN library, only.

### 6.1 Implementation details

The implemented prototype consists of a backend server and a frontend user interface. The backend server is a web application based on Node.js<sup>18</sup> and jQuery<sup>19</sup> and was packaged by NW.js<sup>20</sup> to become a native application. It serves different purposes: a) it maintains and exposes the XDN library with its methods and classes, b) it hosts the predefined *simulated notification sets* and the *predefined simulated device sets*, c) it provides the methods needed to load and/or store developer-defined scripts, device sets and notifications, and d) it provides the methods used by the GUI to simulate the arrival of a notification.

A developer interacts with the frontend application which implements the XDN GUI. The GUI is composed of an editor, and a simulator which includes a log and two modules for loading the device sets and the notifications. The frontend application was built upon ace<sup>21</sup>, an embeddable code editor. When a developer edits the script algorithm, after pressing the “Run code” button, the frontend app automatically

<sup>18</sup> <https://nodejs.org/>, last visited on January 15, 2017

<sup>19</sup> <https://jquery.com/>, last visited on January 15, 2017

<sup>20</sup> <https://nwjs.io/>, last visited on January 15, 2017

<sup>21</sup> <https://ace.c9.io/>, last visited on January 15, 2017

updates the stored code and then interprets the developer's code executing the specified operations. In the current prototypical implementation, the simulated runtime environment adopts JavaScript *eval()* function to interpret developer's code.

Finally, the *x<sub>dn</sub>* class was implemented as a JavaScript object literal providing the notification, device and log objects.

## 6.2 Scenario 1: Messaging Application

This scenario presents the details of the notification strategy developed for the “Messaging Application” scenario presented in Section 3. We have already presented three sample situations, and we have already presented the general rules designed by Ashley. In the following description, we detail such rules.

Every time a new notification is received, the following verifications and consequent actions will be performed by the notification strategy.

- If the two words “@important” and “@personal” are simultaneously present in the generator field of the notification and the user is working (i.e., no Smart TV and Hi-Fi and car Hi-Fi is available, while at least one PC is available): a) the notification content will be shown on all the available PCs and tablets, and b) the vibration will be activated only once on all the available devices that support it;
- If the user is relaxing in her house and in front of her smart TV (i.e., the status of a presence detector installed near the TV reveals the user presence, or one smart TV or one smart Hi-Fi system is available and is playing music/video), the notification content should be shown on all the available TVs. In addition, the vibration will be activated on all the available devices that support it (in case she is inattentive);
- If the user is driving (i.e., a car Hi-Fi system is available and playing music), the system should distinguish useful and useless messages. It is supposed that the generator of the message inserted the word “@unimportant” in the generator field of the notification if it recognizes that it is useless. If the message is useless, only the notification content will be shown on all the available smartphones without any other action. Otherwise, the following actions will be performed: a) the notification content will be shown on all the available smartphones, b) the vibration will be activated on all the available smartphones, c) the notification content will be read on the car Hi-fi system.
- In all the cases that are not covered by the previous ones, a) a textual notification will be shown on all the available smartphones, b) the vibration will be activated on all the available smartphones, c) the smartphones LED will be made blink.

The complete pseudocode of such an application is shown in Listing 2. The authors of this paper employed 18 minutes to implement it. The editor provided by the GUI helped them to immediately identify syntax errors: some red error symbols were shown near the lines in which the errors were. Moreover, the log section and the graphic simulator helped them in identifying the runtime errors, too. The final version of the implemented notification strategy amounts to just 33 lines of code.

### 6.3 Scenario 2: Ambient assisted living

In addition, we developed a second scenario, designed so to cover all the requirements discussed in section 4.

Peter is developing an IoT system to monitor the health of elderly people. The system is able to monitor the elderly behavior, their health status, the status of the house, and can autonomously generate notifications to warn their relatives about unexpected situations. Peter is designing the algorithm that should manage all the notifications arriving from the monitoring devices and wants to test it in a real situation.

To better design such a system, Peter identified a simplified use case that would help him in designing the notification strategies. In this use case only one elderly person (Mary) is considered with her daughter, Julie (they live in the same house). Julie owns two smartphones, one running iOS and another one running Android, a smartwatch, a smart TV, two smart lights, a smart fridge and an Hi-fi connected to the internet. All these devices are able to show and/or loudly read incoming notifications.

The system should generate three different kinds of notifications:

- *dangerous*;
- *warning*;
- *silent*.

The *dangerous notification* aims at warning the user by using all the available devices. This kind of notifications could be sent, for instance when the system recognizes that Mary fell and, consequently, her daughter should be warned as soon as possible. Every available device will be involved: the smart lights will be turned on and will change their color to red, the smart TVs will show the content of the notification playing a warning sound, and all the tablets, PCs, smartphones and smartwatches will show the content of the notification also playing a sound and vibrating.

The *warning notification*, on the other hand, aims at warning the user without scaring her but being sure the message can be easily delivered. Consequently, the smartphone will be the preferred device for showing the notification, but other actions will be used in addition to the visualization of the notification content: a warning sound will be played, in addition to the vibration and the LED blink. Furthermore, to be sure that the user will be informed about the notification, if at least one smartphone is playing some other sounds (so the user is already disrupted by something else), another available personal device (e.g., the smartwatch) will be selected as recipient, too.

Finally, the *silent notification* aims at informing the user but without causing disruption. For example, if the monitored person has just measured her body temperature and all is ok, even though her daughter wants to be informed about it, it is not necessary to attract her attention. Therefore, the smartphone will remain the preferred device for this notification but without any disruptive method (e.g., no vibration nor sound should be played): the content of the notification will be shown on all the available smartphones that have the display on. Otherwise, if no smartphones have the display on, the notification content will be shown on all the available smartwatches.

The full code implemented to manage the described system is composed of 39 lines of code (space limitations prevent us to include the full code in the paper) and



the authors of the paper employed 24 minutes to implement it. As in the previous scenario, the editor provided by the GUI helped authors to identify syntax errors while the log section and the graphic simulator helped them in identifying the runtime ones.

In all the presented applications, XDN helped the developers in implementing the code to customize and distribute cross-device notifications. In general, we observed that most of the time was needed to differentiate the designed situations and the corresponding behaviors of the devices based on their status and/or the properties of the received notification.

## 7 Discussion

This section presents a preliminary analysis of the actual contribution brought by the XDN framework in supporting developers in designing algorithms able to customize and distribute cross-device notifications.

Starting from the observation, supported by related works, about the lack of support for the customization of notifications and the development of cross-device notification strategies, the emphasis of the analysis has been put on the actual advantages and challenges that the XDN framework could provide for developers. This analysis was conducted through the development of the two scenarios presented in the previous section and it was performed by the authors of the paper.

### 7.1 Successes

In Section 4, seven different requirements were presented.

The first one regarded the need of APIs to support developers in: a) selecting available mobile/IoT devices based on their specific properties and status, b) performing specific actions on selected devices (e.g., turn on a LED or make it blink), and c) manage notification properties (e.g., arrival time, notification content). We can claim that the methods and classes provided by the XDN library in conjunction with the services provided by the XDN runtime environment satisfy R1.

In addition, R2 regarded the need of a graphical interface. The designed GUI is able to satisfy all the low-level described requirements: helping developers in implementing the notifications strategies' scripts, b) visualizing the current status of available devices, c) visualizing/simulating the behavior of the designed algorithms/strategies.

Furthermore, the presence of an editor, a log, a module to load devices' portfolios and statuses, a module to load notifications and the possibility of running the simulation satisfy R3 and R4 requirements regarding the necessity of a device simulator that is also able to simulate more than one devices at the same time.

The use of JavaScript, one of the most commonly used programming languages, satisfies R7 and makes XDN be compatible with Chord. Moreover, the use of JavaScript merged with the presence of the XDN runtime environment and the presence

of the XDN IoT/mobile library also satisfies R5 that regarded the possibility of distributing notification strategies among multiple platforms. The spread of such programming language, in fact, guarantees a high compatibility with most of existing mobile and IoT systems and motivate developers in using the XDN framework for their applications.

Finally, R6, that regarded the support for IoT devices, is satisfied by the two modules that allow to load and manage IoT devices and their properties.

## 7.2 Challenges

Some challenges were identified while using this preliminary implementation of the XDN framework for developing the notification strategies described in the scenarios.

The first challenge regards the XDN GUI and specifically, the editor. Even though, in this initial implementation, the editor recognizes JavaScript syntax errors, it is not yet able to recognize errors in using the XDN API. For example, if the developer writes “DeviceType” instead of “deviceType” the editor does not recognize it as an error. Although the log helps developers identifying such errors, such a feature would be really appreciated by developers.

Furthermore, another challenge regards the interaction with the available devices. With the current version of the XDN framework it is possible to customize notification strategies and graphically simulate the behavior of the devices when a new notification arrives. However, the current implementation does not treat the possibility of performing specific actions due to the reaction of users to notifications. So, for example, it is not possible to capture the user disruption to notifications and, in consequence, implement some extra code to perform new actions after a predefined time from the one at which, for instance, the user swiped a notification away.

Finally, the last challenge regards the absence of the XDN runtime environment in the developed prototype: even if the simulator helped the authors of the paper in testing their notification strategies, the lack of the XDN runtime environment did not allow them to test their strategies in the wild with real devices.

## 8 Conclusions

This paper proposed a framework for developers to create and distribute cross-device notifications by scripting. The XDN architecture includes a) an XDN library to assist developers in designing personalized notifications to be distributed among ad-hoc networks of mobile and IoT devices, b) an XDN GUI to assist developers in implementing notification strategies and testing them by simulating the arrival of notifications, c) an XDN runtime environment for receiving notifications from IoT/mobile devices, executing the deployed notification strategies, and sending commands to be executed on the devices, and d) an XDN IoT/mobile library to both generate notifications compatible with XDN and execute the commands received by the XDN runtime environment.

Using one simple scenario as a running example, the major components of the framework were presented and explained. To demonstrate the feasibility of the framework and the fulfillment of all the presented requirements, two different realistic scenarios were designed and implemented. During the assessment the emphasis was also put on the time needed to implement the desired applications and on the number of lines of code needed to implement the desired behaviors: results demonstrate that XDN framework is a promising technology.

Assessment revealed some challenges that will be addressed in future works. As a future work, the framework will be, also, evaluated in one or more complete user studies, with groups of developers. In addition, the prototype will be enhanced by adding other functions that developers may suggest during test sessions.

## References

1. Adamczyk, P.D., Bailey, B.P.: If not now, when?: The effects of interruption at different moments within task execution. In: Proc. SIGCHI Conference on Human Factors in Computing Systems, CHI '04, pp. 271–278. ACM (2004)
2. Arlein, R.M., Betg -Brezetz, S., Ensor, J.R.: Adaptive notification framework for converged environments. Bell Labs Technical Journal **13**(2), 155–159 (2008)
3. Bailey, B.P., Konstan, J.A.: On the need for attention-aware systems: Measuring effects of interruption on task performance, error rate, and affective state. Computers in Human Behavior **22**(4), 685 – 708 (2006). Attention aware systemsSpecial issue: Attention aware systems
4. Corno, F., De Russis, L., Montanaro, T.: A context and user aware smart notification system. In: 2015 IEEE 2nd World Forum on Internet of Things (WF-IoT), pp. 645–651 (2015)
5. Hamilton, P., Wigdor, D.J.: Conductor: Enabling and understanding cross-device interaction. In: Proc. SIGCHI Conference on Human Factors in Computing Systems, CHI '14, pp. 2773–2782. ACM, New York, NY, USA (2014)
6. Horvitz, E., Kadie, C., Paek, T., Hovel, D.: Models of attention in computing and communication: From principles to applications. Commun. ACM **46**(3), 52–59 (2003)
7. Husmann, M., Spiegel, M., Murolo, A., Norrie, M.C.: Ui testing cross-device applications. In: Proc. 2016 ACM on Interactive Surfaces and Spaces, pp. 179–188. ACM, New York, NY, USA (2016)
8. Iqbal, S.T., Bailey, B.P.: Effects of intelligent notification management on users and their tasks. In: Proc. SIGCHI Conference on Human Factors in Computing Systems, CHI '08, pp. 93–102. ACM, New York, NY, USA (2008)
9. Iqbal, S.T., Horvitz, E.: Notifications and awareness: A field study of alert usage and preferences. In: Proc. 2010 ACM Conference on Computer Supported Cooperative Work, CSCW '10, pp. 27–30. ACM, New York, NY, USA (2010)
10. Koss, M.C., Dewitt, J., Messerly, K.J., Titov, D.: Cross-device notifications (2015). US Patent 2015/0373089
11. Kubitz, T., Voit, A., Weber, D., Schmidt, A.: An IoT infrastructure for ubiquitous notifications in intelligent living environments. In: Proc. 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing, UbiComp '16, pp. 1536–1541. ACM, New York, NY, USA (2016)
12. Kushlev, K., Proulx, J., Dunn, E.W.: "silence your phones": Smartphone notifications increase inattention and hyperactivity symptoms. In: Proc. 2016 CHI Conference on Human Factors in Computing Systems, CHI '16, pp. 1011–1020. ACM, New York, NY, USA (2016)
13. Mehrotra, A., Hendley, R., Musolesi, M.: Prefminer: Mining user's preferences for intelligent mobile notification management. In: Proc. 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing, UbiComp '16, pp. 1223–1234. ACM, New York, NY, USA (2016)
14. Mehrotra, A., Musolesi, M., Hendley, R., Pejovic, V.: Designing content-driven intelligent notification mechanisms for mobile applications. In: Proc. 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing, UbiComp '15, pp. 813–824. ACM, New York, NY, USA (2015)
15. Mehrotra, A., Pejovic, V., Vermeulen, J., Hendley, R., Musolesi, M.: My phone and me: Understanding people's receptivity to mobile notifications. In: Proc. 2016 CHI Conference on Human Factors in Computing Systems, CHI '16, pp. 1021–1032. ACM, New York, NY, USA (2016)

16. Nebeling, M., Husmann, M., Zimmerli, C., Valente, G., Norrie, M.C.: Xdsession: Integrated development and testing of cross-device applications. In: Proc. 7th ACM SIGCHI Symposium on Engineering Interactive Computing Systems, EICS '15, pp. 22–27. ACM, New York, NY, USA (2015)
17. Schreiner, M., Rädle, R., Jetter, H.C., Reiterer, H.: Connichiwa: A framework for cross-device web applications. In: Proc. 33rd Annual ACM Conference Extended Abstracts on Human Factors in Computing Systems, CHI EA '15, pp. 2163–2168. ACM, New York, NY, USA (2015)
18. Seshadri, P., Abileah, S., Nilakantan, N., Knight, H., Pather, S., Gerber, R., Mensa-Annan, C., Garrett, P., Faoro, M., Lavery, D.: User interface system and methods for providing notification(s) (2008). US Patent 7,360,202
19. Weber, D., Shirazi, A.S., Henze, N.: Towards smart notifications using research in the large. In: Proc. 17th International Conference on Human-Computer Interaction with Mobile Devices and Services, MobileHCI '15, pp. 1117–1122. ACM, New York, NY, USA (2015)
20. Weber, D., Voit, A., Kratzer, P., Henze, N.: In-situ investigation of notifications in multi-device environments. In: Proc. 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing, UbiComp '16, pp. 1259–1264. ACM, New York, NY, USA (2016)
21. Yang, J., Wigdor, D.: Panelrama: Enabling easy specification of cross-device web applications. In: Proc. SIGCHI Conference on Human Factors in Computing Systems, CHI '14, pp. 2783–2792. ACM, New York, NY, USA (2014)
22. Yuan, F., Gao, X., Lindqvist, J.: How busy are you?: Predicting the interruptibility intensity of mobile users. In: Proc. 2017 CHI Conference on Human Factors in Computing Systems, CHI '17, pp. 5346–5360. ACM, New York, NY, USA (2017)