XDN: Cross-Device Framework for Custom Notifications Management

XDN: Cross-Device Framework for Custom Notifications Management

Fulvio Corno
Politecnico di Torino
Corso Duca degli Abruzzi, 24
Torino, Italy 10129
fulvio.corno@polito.it

Luigi De Russis
Politecnico di Torino
Corso Duca degli Abruzzi, 24
Torino, Italy 10129
luigi.derussis@polito.it

Teodoro Montanaro
Politecnico di Torino
Corso Duca degli Abruzzi, 24
Torino, Italy 10129
teodoro.montanaro@polito.it

ABSTRACT

As notifications become part of people's lives, their importance often depends on various factors that can influence the reaction and the disruption of recipients. The generation and the distribution of notifications should be carefully designed every time a new application or smart device is devised. This paper presents XDN (Cross Device Notification), a framework to assist developers in creating cross-device notifications by scripting. XDN provides a set of high-level APIs, based on JavaScript, for designing personalized notifications to be distributed among ad-hoc sets of end-user devices. Developers are also supported in implementing and testing notification strategies thanks to an integrated environment. We present a use case to demonstrate the functionality and the applicability of the framework.

CCS CONCEPTS

• **Human-centered computing** → **Interactive systems and tools**; • **Software and its engineering** → *Development frameworks and environments*;

KEYWORDS

Cross-Device, Framework, Notifications, Developer, API

ACM Reference format:
Fulvio Corno, Luigi De Russis, and Teodoro Montanaro. 2017. XDN: Cross-Device Framework for Custom Notifications Management. In *Proceedings of EICS '17, Lisbon, Portugal, June 26-29, 2017,* 6 pages.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

EICS '17, June 26-29, 2017, Lisbon, Portugal

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

During the last decade, the presence of notifications in people's routines has grown, with the aim of facilitating their lives. Although people are becoming accustomed to notifications, the usefulness and the importance of each notification often depends on various factors that can influence the reaction and the disruption of recipients. According to Seshadri et al. [6], 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". For this reason, the influence brought by notifications on users should be taken into account by developers in order to "deliver timely and personalized information on whatever suitable device is available and accessible" [6]. In this way developers will be able to increase the efficacy of both the generation and the distribution of notifications and bring, at the same time, a consequent alteration of user reaction (e.g., a reduction of the recipient annoyance). Furthermore, the Internet of Things (IoT) is also gaining importance in the notification context. The growing spread of the IoT is introducing new devices every day and "the ongoing wave of smart devices makes it possible to reach the user through multiple devices at once, amplifying the effects of notifications" [7]. Even though it is necessary to personalize notifications to differently handle important notifications and unimportant ones, it is also important to develop strategies able to transform the disadvantages of receiving the same notification on multiple devices into advantages. A contribution in this challenge could be brought by the application of the cross-device approach [2] to notifications, i.e., extending an application user experience across multiple devices.

This paper presents XDN (Cross Device Notification), a framework that assists developers in designing personalized notifications to be distributed among ad-hoc networks of mobile and/or IoT devices. With XDN, developers can explore and evaluate different design alternatives at reduced cost and time. The framework is composed of two main parts: 1) the XDN library that implements a set of high-level APIs to create

cross-device JavaScript algorithms and rapidly generate and distribute cross-device notifications among available mobile and/or IoT devices; and 2) the XDN GUI that provides a) an IDE for allowing developers implement and test their algorithms and b) a simulator that shows devices behavior when a new notification arrives.

2 RELATED WORKS

Framework for customizing or distributing notifications

Notifications have been extensively treated in the literature, but only a few works provide developers with the possibility of customizing notifications and/or their distribution. An interesting work able to smartly distribute notifications among different IoT devices is proposed by Kubitza et al. [5]. They present an infrastructure for homes and offices that enables developers to design and deploy context sensitive notification strategies using arbitrary things and smart home products connected to their meSchHub gateway. In the proposed infrastructure, the notifications received on the user smartphone are sent to the meSchHub gateway that forwards them according to interaction scripts pre-defined by designers. According to the provided description, the meSchHub system lacks, at first, a support for cross-device distribution of notifications. Lastly, it lacks a simulator: developers are not allowed to test their interactions scripts on devices that are not physically owned. 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. [6] that presents a system and a methodology to facilitate the development, debug, and deployment of a notification platform application. According to their description, the system only allows the interaction with configuration files, without providing a GUI for simulating the designed algorithms. In addition, it does not support neither any cross-device interaction, nor IoT devices.

In addition to the solutions found in literature, developers can use various existing commercial frameworks and/or APIs for the development of mobile/IoT applications. As an example, Apple provides the UserNotifications and the UserNotificationsUI frameworks to let developers customize notifications in their applications, and Google provides a similar solution for Android devices. However, some drawbacks emerge by analyzing the documentation provided for both solutions. First of all, it is not currently possible to develop cross-device strategies for distributing notifications. Second, apart from few supported smart TVs, these solutions do not consider other IoT devices. Third, they are strictly limited to the development of applications for specific platforms. Thus, if a notification strategy is developed for a platform (e.g., Android) it is not easy to export and use it in other platforms. Finally, they do not allow the simultaneous simulation of different devices to test the designed notification strategy on all the target devices.

Cross-device notifications

The development of cross-device applications has already been applied in different domains to solve different problems (e.g., [1, 2]), but only a limited number of works are devoted at the generation and/or the cross-device distribution of notifications among different devices. Horvitz et al. [3] present the Notification Platform, a cross-device messaging system that modulates the flow of messages from multiple sources to other devices by performing ongoing decision analyses. 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. Campbell et al. [4], instead, present some techniques for cross-device notifications. Authors start from the consideration that a notification could be missed due to any reason (e.g., because the smartphone is in a bag) and, even though other devices are in use, the user remains unaware about it. They propose a solution that involves available devices to allow user to be warned about incoming notifications. The main contribution of presented works is related to the possibility of distributing cross-device notifications among mobile devices. In our work, we embrace the same approach but we extend it to IoT devices. Likewise, one of the most interesting works that support cross-device interactions among mobile devices is the Chord [1] framework. 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. Chord is mainly designed to assist the implementation of cross-device interactions and it does not give any support for notifications nor for IoT devices. However, the cross-device nature of Chord, its ease of use, and its linear data structure, guided us to develop XDN to be compatible with Chord so that it could be possibly integrated as a future extension.

3 REQUIREMENTS

We identified two high-level requirements for a cross-device notifications framework. The first one (API, R1-R3) aims at reducing repetitive code and encouraging rapid development of algorithms for customizing and distributing notifications. The latter (GUI, R4-R7) regards the need of having a graphical

interface to develop and then test such algorithms in an environment able to simulate the arrival of one or more notifications.

R1. Access to notifications content One of the most complete cross-device solution for developers found in literature is Chord. However, it is not natively able to manage notifications, their properties, and all the specific mobile/IoT devices' features that could be used by developers to warn the user about the arrival of a notification (e.g., turn on a LED or make it blink). Consequently, XDN should provide developers with the possibility of accessing notifications' content, together with other related information: date and time of receipt, generator and, if available, icon.

R2. Multi-platform Although different existing solutions (e.g., development of an app through Android Studio) let developers customize notifications and/or their distribution, they mainly allow the creation of software that is limited to some specific devices (e.g., Android devices). XDN should allow the development of scalable notification strategies that could be easily exported on different platforms.

R3. Support for IoT devices Nowadays, with the increasing spread of the IoT, new smart devices and appliances are developed everyday with the ability to show notifications. Consequently in addition to existing mobile devices, the XDN framework should support existing IoT devices.

R4. Editor With the aim of supporting developers in implementing their algorithms using the XDN APIs, the XDN framework should provide a graphical editor that guides developers in the correction of programming errors.

R5. Support for loading predefined use cases XDN should provide some sample scenarios composed by different devices, their properties, and their corresponding current statuses (i.e., the current values assumed by each property). The devices' properties (as defined in a device portfolio) and the devices' statuses should be provided separately, so that the same device could have different statuses for the tests.

R6. Simulator XDN should provide a simulator able to simulate the arrival of a notification. The simulator should graphically show the behavior of predefined devices running the defined code.

R7. Notifications definition XDN should provide an easy way to load "test" notifications from existing developer-defined external files. In addition, some predefined notifications could be provided in the GUI to be used as examples.

4 USE CASE: STRATEGY VIDEOGAME

Before presenting the XDN framework, we introduce a use case that will be used as an example in the next sections.
John is developing a strategy videogame for smartphones and he wants to specify different behaviors for his game. The behaviors mainly depends on two different kinds of information that can be generated: the INFO notifications should inform the user about the progresses of her virtual world, and the WARN notifications should inform the user about the necessity of his intervention in the game. John designed the system to select all the smartphones as recipients of an "INFO" notification. The notification content will be shown on all the smartphones and the smartphones led will be made blink with green color. For the "WARN" notifications, instead, John decided that smartwatches will be the preferred devices for informing the user. However, if all the smartwatches have the volume to 0, the smartphones will be used to play a warning sound.

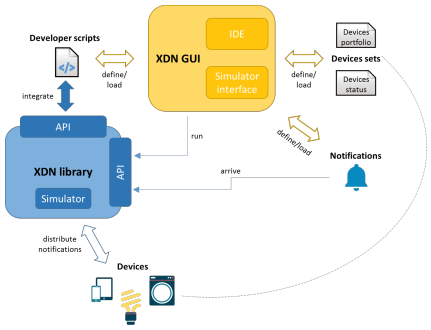

Figure 1: High level blocks that compose the XDN framework

5 FRAMEWORK

The architecture of the framework (Figure 1) is composed of two main blocks: the XDN library and the XDN GUI. The XDN library provides all the classes, methods, and objects to facilitate the creation of algorithms able to customize and distribute cross-device notifications. In addition, it provides all the methods used by the GUI to simulate the behavior of the devices while new notifications arrive. On the other hand, the XDN GUI provides all the graphical tools to easily interact with the XDN library. The XDN GUI, in fact, provides a web-based user interface that allows developers to:

- write their JavaScript algorithms or load them from an existing developer script;
- define or load a devices set to be used during the simulations;
- define or load a list of notifications used during the simulation as arriving notifications;
- run the simulation, to simulate the arrival of notifications showing the behavior of all the available devices;
- visualize the behavior of all the loaded devices during the simulation.

A devices set is composed of some devices' properties (defined in a devices portfolio) that are static and represent the

device capabilities, and the corresponding current devices' statuses, that represent the current values assumed by each device's property.

Framework GUI. John, the developer of our use case, wants to develop a strategy videogame and, after the design phase, he decides to use the XDN framework to implement the notification manager of his game. As a first step, he connects to the web-based XDN GUI (Figure 2) and, looking at the documentation, he starts from the selection of the devices a user will supposedly use during the gameplay. He can choose between two options: select one of the existing set of devices provided in the XDN GUI or create a new set of devices by himself. He decides to choose the first option and he selects and loads the predefined set of devices (i.e., a smartphone and a smartwatch). He is sure that all the devices were successfully loaded because they appeared in the column of simulated devices (column "Device Set", letter B in Figure 2). Now it is time to write the code he will integrate in his application. He can do it by using the editor present in the GUI, located in the left column (letter A). After creating the algorithm, John wants to test it using the simulator included in the XDN framework. He has to perform three actions: a) load the statuses of the available devices by using the buttons present at the top of column marked with letter B, b) load the list of notifications with which he wants to test the algorithm, by using the buttons present at the top of the column marked with letter C and, c) run the simulation using the *Run code* button located at the top of the editor (letter A). When all these actions are performed, John will see all the updated statuses in the central column dedicated to simulated available devices (letter B) and will be able to analyze the list of all the performed actions (in sequence) in the log section (letter D).

Framework API. The XDN APIs provide a set of clearly defined methods that allow developers to customize and distribute cross-device notifications by reducing repetitive code. They are based on two main objects: xdn.notification and xdn.device.

xdn.notification implements the *Notification* sub-object and provides the *notification handler interface*. The *Notification* sub-object represents a single notification and contains all the properties reported in the Table 1. Instead, the *notification handler interface* is responsible for intercepting the incoming notifications with the aim of performing the action specified by the developer in the handler implementation. Consequently, all the code implemented by John to customize and distribute arriving notifications should be written as a notification handler implementation. A snippet of the code needed by John to log the content of the received notification is shown in Listing 1, as an example.

Table 1: Notification properties

Property	Type
dateTime	date and time of notification receipt
content	the content of the notification
generator	the generator of the notification
icon	the optional icon associated to the notification

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

Listing 1: Log notification content

The **xdn.device** object implements all the classes, sub-objects and methods needed by developers to interact with available devices. The two main sub-objects provided by the xdn.device objects are *Device* and *DeviceSelection*. The *Device* sub-object represents a single device and contains all its properties and all its statuses. Instead, the *DeviceSelection* sub-object contains all the methods and functions applicable for a) searching and filtering across a set of devices, and b) perform actions on one or more selected devices.

The *Device* properties can be divided between static properties and the corresponding statuses: even though the framework is designed to import them separately (the XDN GUI asks separately for a set of devices' portfolios, the static properties and a set of devices' statuses), they are treated as a unique object and contained in each instance of "*Device*". Each property will be assigned to each device depending on its nature. For example, if John is using a tablet, it will have for sure a display and a speaker, consequently the properties *display* and *speaker* will be specified in the *Device* object. In addition, these properties could have one or more sub-properties (e.g., a display has a *size* sub-property). All the possible properties and sub-properties of a device are listed in Table 2 in the *Property* and *Sub-Property* columns. Instead, all the status properties and sub-properties are listed in Table 3. In both tables, *name* is the unique identifier.

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

- *xdn.device.select*: returns a list of Device objects that satisfy the specified criteria (e.g., the code `xdn.device.select('deviceType=="smartwatch"')` returns all the smartwatches);
- *xdn.device.selectWith*: returns a list of Device objects that has the specified property (the property is set);
- *xdn.device.selectAll*: gets all the devices;

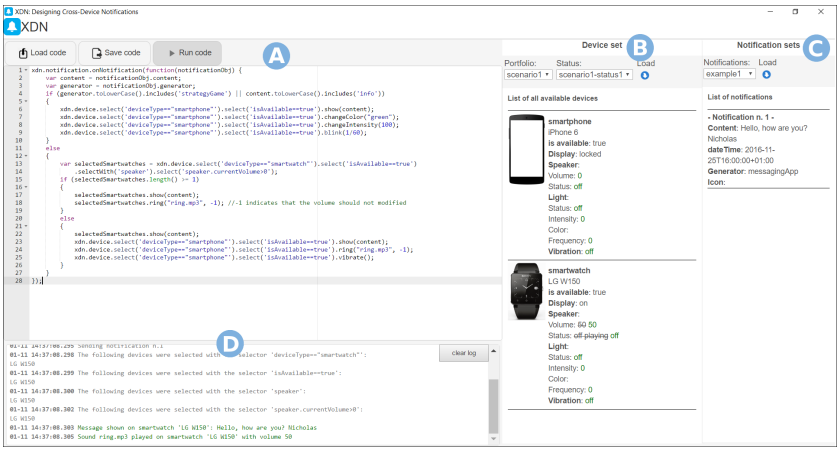

Figure 2: Screenshot of the XDN GUI

Table 2: Device properties

Property	SubProperty
name	-
deviceType	[smartphone, smartwatch, bracelet, smartLight, tablet, PC, fridge, hi-fi, smartTv, carHi-fi]
display	size, privacy: [high, normal, low], touch: [true, false]
speaker	privacy: [high, normal, low]
light	colors, intensity: [true, false], frequency: [true, false], blink: [true, false]
vibration	[true, false]
os	-

Table 3: Device statuses

Property	SubProperty
name	-
isAvailable	[true, false]
display	currentStatus: [on, locked, off]
speaker	currentVolume, currentStatus: [playing, off]
light	currentStatus: [on, off, blinking], currentIntensity, currentColor
vibration	currentStatus

- *xdn.device.getDeviceName*: returns the Device object with the specified name (*deviceName* property set to the specified name)
- *xdn.device.not*: returns all the Device objects that does not satisfy the specified criteria. This method is used to exclude one or more devices, e.g., if John wants to exclude all the smartwatches: `xdn.device.not('deviceType=="smartwatch"')`.

Table 4: Device actions

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

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 John wants to select all the smartphones that are available he can use the following snippet:

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

Table 4 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 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. The final algorithm written by John to implement the designed behavior of his strategy game is available at https://elite.polito.it/files/xdn/strategyGame.js.

6 IMPLEMENTATION

The XDN framework consists of both a backend server and a frontend user interface. The backend server is a web application based on Node.js and jQuery and was packaged by

NW.js[1] (previously known as node-webkit) 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 Device sets and Notifications' samples defined in JSON format, 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 includes the web IDE and the simulator interface. The frontend application was built upon ace[2], an embeddable code editor written in JavaScript, jQuery, and Bootstrap.

7 PRELIMINARY EVALUATION AND DISCUSSION

The emphasis of the preliminary analysis has been put on the actual advantages and disadvantages that the XDN framework could provide for developers. The analysis was conducted through the development of the "Strategy Videogame" use case and it was performed by the authors of the paper. Specifically, the use case was implemented using the XDN GUI in 12 minutes and only 21 lines of code were necessary to implement it. In addition, writing code to select devices or to access notification content took only a few lines of code and a few minutes. Instead, more time was needed to differentiate the different behaviors required by the use case.

In the *Requirements* section, two different high-level requirements were presented. The first one regarded the development of a library able to reduce repetitive code and encourage rapid development of algorithms aimed at customizing and distributing notifications. Looking at the lines of code needed to develop the described use case reported, we can claim that this first requirement was satisfied. If, in fact, we could recreate the same notification strategies without using XDN, we would implement all the code needed to select all the devices, check for the existence of the properties, check for the current statuses, and finally select only the devices that satisfy the specified criteria. Instead, by using XDN, only one line of code was needed to do all the described procedures, and only a few other lines of code were necessary to perform the desired actions on the selected devices. Furthermore, we can claim that the designed GUI is able to satisfy all the low-level requirements of the second high-level requirement (GUI). 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 could, in fact, motivate developers in using the framework for their applications.

Finally, we identified a challenge regarding the XDN API and, specifically, the two *Device* and *DeviceSelection* objects. JavaScript developers are typically accustomed to use objects and their properties through variables. However, XDN does not allow a direct access to devices and their statuses: it is necessary to use the *DeviceSelection* object and its methods to access, filter, search and/or drive them. This abstraction could be difficult to understand by developers, but it is necessary for the future enhancements in which real devices will be driven through the framework.

[1]https://nwjs.io/, last visited on January 15, 2017

[2]https://ace.c9.io/, last visited on January 15, 2017

8 CONCLUSIONS

This paper proposed a framework for developers to create and distribute cross-device notifications by scripting. XDN provides a) a set of high-level APIs and b) a web-based integrated GUI for creating algorithms able to manage cross-device notifications. The GUI also allows the simulation of the devices' behavior when new notifications arrive. By using a use case as a running example, all the components of the framework were presented and explained. To demonstrate the functionality and the applicability of the framework, the proposed use case was actually implemented. The evaluation revealed some challenges that will be addressed in future works, together with a more complete evaluation with users.

REFERENCES

[1] Pei-Yu (Peggy) Chi and Yang Li. 2015. Weave: Scripting Cross-Device Wearable Interaction. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems (CHI '15)*. ACM, 3923–3932. https://doi.org/10.1145/2702123.2702451

[2] Peter Hamilton and Daniel J. Wigdor. 2014. Conductor: Enabling and Understanding Cross-device Interaction. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '14)*. ACM, New York, NY, USA, 2773–2782. https://doi.org/10.1145/2556288.2557170

[3] Eric Horvitz, Carl Kadie, Tim Paek, and David Hovel. 2003. Models of Attention in Computing and Communication: From Principles to Applications. *Commun. ACM* 46, 3 (March 2003), 52–59. https://doi.org/10.1145/636772.636798

[4] M. C. Koss, J. Dewitt, K. J. Messerly, and D. Titov. 2015. Cross-Device Notifications. (December 2015). US Patent US 2015/0373089.

[5] Thomas Kubitza, Alexandra Voit, Dominik Weber, and Albrecht Schmidt. 2016. An IoT Infrastructure for Ubiquitous Notifications in Intelligent Living Environments. In *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct (UbiComp '16)*. ACM, New York, NY, USA, 1536–1541. https://doi.org/10.1145/2968219.2968545

[6] P. Seshadri, S. Abileah, N. Nilakantan, H. Knight, S. Pather, R.H. Gerber, C.T. Mensa-Annan, P. Garrett, M.A. Faoro, and D.O. Lavery. 2008. User interface system and methods for providing notification(s). (April 2008). US Patent 7,360,202.

[7] Dominik Weber, Alireza Sahami Shirazi, and Niels Henze. 2015. Towards Smart Notifications Using Research in the Large. In *Proceedings of the 17th International Conference on Human-Computer Interaction with Mobile Devices and Services Adjunct (MobileHCI '15)*. ACM, New York, NY, USA, 1117–1122. https://doi.org/10.1145/2786567.2794334