ScreenPress: A Powerful and Flexible Platform for Networked Pervasive Display Systems

Amir E. Sarabadani Tafreshi and Moira C. Norrie
Department of Computer Science, ETH Zurich
CH-8092 Zurich, Switzerland,
{tafreshi | norrie}@inf.ethz.ch

ABSTRACT
Developing a pervasive display system (PDS) tailored to the requirements of a particular environment in terms of the set of devices available, content curation, visualisation, modes of interaction and user customisation can be non-trivial. A platform general enough to support the wide variety of PDSs of current interest to both researchers and display owners is still missing. Based on an analysis of a wide variety of existing systems in research and industry, we propose a general metamodel for PDSs and present an implementation based on the popular content management system WordPress. We demonstrate the flexibility and use of the developed platform (ScreenPress) by describing the process of developing a showcase application that integrates a variety of services. In addition, we report on a user study conducted with twelve participants with experience in managing the content of various PDSs.

ACM Classification Keywords
D.2.2 Software Engineering: Design Tools and Techniques

Author Keywords
PDS requirements; content management; public displays.

INTRODUCTION
With the widespread deployment of public and semi-public screens, it is now common for all sorts of organisations to use some form of pervasive display system (PDS). Researchers have actively investigated potential applications and technology solutions to support global PDS networks, often using various sensors and devices to turn those into interactive displays. Consequently, the complexity of PDSs has increased significantly and new requirements have arisen.

A PDS should not only be easy to tailor to an organisation's requirements, such as customising the content to be displayed and its visualisation, but also easy to operate. A general PDS platform therefore has to support the configuration and management of PDSs as well as simple and convenient means for content creation and acquisition.

PerDis '17, June 07-09, 2017, Lugano, Switzerland
 ISBN 978-1-4503-5045-7/17/06…15.00
DOI: http://dx.doi.org/10.1145/3078810.3078813

At the same time, researchers are continuing to explore new forms of PDSs that may involve personal and/or public screens and be novel in terms of the applications, screen technologies, visualisations and/or modes of interaction. For these new PDSs, as well as existing systems, they are also interested in studying the impact on users and user communities. Researchers therefore want a platform that can support the rapid prototyping of a wide range of scenarios and is easy to extend with new services, visualisations, modes of interaction and forms of communication. Although a number of PDS toolkits and frameworks have been proposed in the past, they tend to either be limited in the types of applications they support or require non-trivial programming effort to develop an application.

To address the need for more powerful and flexible platforms to support the wide variety of modern and emerging PDSs, we first developed a general metamodel for PDSs based on an analysis of requirements derived from a study of the research literature on PDSs along with existing systems. Since many of these requirements are partially or fully met by modern content management systems (CMS), we decided to investigate how a general platform for PDSs centred on our model could be implemented on top of WordPress[1], which is currently the most widely used CMS. Our goal was that it should be possible for an unskilled user to set up, adapt and extend different aspects of a PDS, similar to the way in which WordPress enables end-users to set up a website in a few clicks.

BACKGROUND
PDSs are now in everyday use and have become commonplace throughout the workplace and also in public areas. To support the rapid prototyping and evaluation of PDSs, a key requirement is a general and flexible development platform [4]. However, many existing projects either build new applications from scratch [9, 15, 12] or provide a simple management system for a set of fixed applications [10, 18].

Whatever the application and mode of interaction, the choice of content, its source and its visualisation continues to be a major factor [8, 10, 22, 25, 26]. Content curation is the process of discovering, aggregating, selecting and gathering relevant content and then sharing or presenting it to audiences in a targeted and optimised way [21]. Although the benefits of content curation have been recognised, existing PDS tools such as [11, 13, 18] were designed to either directly use the

[1] http://www.wordpress.org

content pulled from third-party resources or create it from scratch without content curation support.

A few platforms used as research testbeds do exist such as [3, 16, 20]. E-Campus is a research testbed that has been used in numerous research projects [3, 7]. The system infrastructure is mainly used for distributed control of displays and content [3]. The roles of content providers and display owners are separated. Content providers make content available as channels to which display owners can subscribe and schedule when content is to be displayed. E-campus also enables the connection of different sensors to each client [6]. However, e-Campus lacks the flexibility to adapt individual content items and visualisations to different contexts since they represent content items simply as files which are scanned and displayed in full screen mode. Also, there is no workflow model that can support some kind of editorial process to ensure that only high-quality content is published.

A more recent real-world system is UBI Hotspot [20]. The system architecture enables content of public displays to be contributed from multiple third parties through the publication of services using a simple URL. In addition to enabling interaction with services on touch-capable large displays, the platform allows application user interfaces to be distributed to personal mobile devices [14]. However, independence of third party applications does not allow curation, full control and management of the system content through a unified tool.

Some researchers have proposed an approach similar to mobile apps for public displays where applications created by third-parties can be accessed via an application store [4]. In comparison with their mobile counterparts, such stores would face specific challenges such as dealing with scheduling requirements, new business models, and multiple stakeholders. Although some application stores, e.g. Mercury [5], have been fully operational for a few years, so far, there are only a few registered applications. This might suggest that, for an application store to be accepted by PDS developers and widely used, it needs to be part of a more general and flexible platform offering developers a wide range of features and facilities.

We propose an alternative approach which is to build on top of a modern CMS such as WordPress, thereby putting content curation at the core. A CMS already offers lots of support for creating, managing and publishing content in web contexts as well as offering some kind of plugin mechanism to support extensibility. This allows various services to be integrated into a single PDS application in contrast to the app approach proposed by other researchers. Consequently, our approach results in a unified interface as well as the ability to configure and manage all aspects of the system through a single tool. Further, by adopting a web-based approach, solutions are not tied to any particular platform or kind of device.

Our aim is to take previous work on frameworks further by developing a platform general enough to support the wide variety of PDSs of current interest to researchers. In particular, we want to cater for the scenarios that involve multiple users and screens, where the content, visualisation and mode of interaction can be customized for both users and devices.

REQUIREMENTS OF A PDS PLATFORM

Based on a systematic study of the literature and various systems, including more recent open display networks [8, 27, 29] as well as the traditional forms of PDSs in use today [6], we derived a general list of requirements for a PDS platform designed to support researchers, developers and end-users. Our questions were (1) what features, functionalities and requirements existing PDS frameworks, platforms, and applications support? (2) what are the challenges for PDSs and how do existing solutions support them? (3) what are the envisioned applications and the requirements to support them? In the literature review, first, relevant papers were identified by analysing publications' titles and abstracts, and in a second stage, a full-text analysis was undertaken to discover and record the requirements. Afterwards, the primary studies were subsequently filtered by keywords.

R1: Content management. Previous research [26, 28] has shown that content is perceived as the system for most PDS users. For example, in e-Campus [26], when a display had no content, viewers assumed that the system was broken.

R2: Separation of content and visualisation. The choice of how content is visualised can vary enormously depending on: the primary goal of the application, the type of content, the location of the display, the set of stakeholders and the business model. Therefore, a PDS should support alternative forms of visualisation ranging from the more traditional ones to ambient displays that visualise information in an abstract way, for example [15].

R3: Support for multi-service applications. It should be possible to use a single PDS for multiple purposes, and hence it needs to be able to serve multiple services [17].

R4: Support for heterogeneous displays. Applications should be able to run on many different types of display and execution environments [29].

R5: Facilities for content curation. Researchers have emphasised the significance of curating the content in attracting and retaining user interest and engagement [19].

R6: Framework for context-awareness. Since contextual factors can be very dynamic across applications [2], content and its visualisation may be adapted according to the context in which a display is situated, including the presence and behaviour of its viewers.

R7: Pervasive sensing and interaction. Interaction with a PDS might increase user engagement [1]. Since the level and forms of interaction are highly variable according to the goals of the application, it is important to support different interaction modalities and forms of context-sensing. In particular, it is important to cater for situations where context sensors may be distributed and even external to the PDS. Further, displays should be able to share context. A PDS platform must therefore also support the possibility of a global sensor network.

R8: Configuration of individual displays. It should be possible to configure displays individually so they can show different content in the same context, or even the same content with different visualisations.

R9: Support for dynamic display networks. A general PDS platform must be able to support a heterogeneous network of displays that is dynamic, flexible and scalable. Further, it must provide simple means of managing the network.

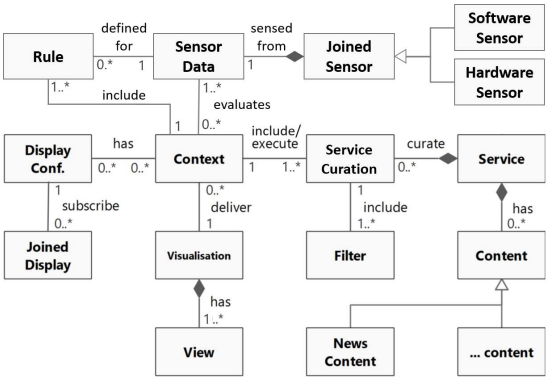

Figure 1. Proposed core model

R10: User management. A PDS has multiple stakeholders including display owners, content providers and system administrators, with differences in roles tending to arise as the size and complexity of a display network increases [4]. A PDS platform must therefore provide facilities for user management that allow role definition and access rights control.

R11: Workflow management. Large display networks may involve a team of content providers, designers, content curators, context experts and display owners and it is important to provide facilities to coordinate their activities.

R12: Update tracker. Since it should be possible to dynamically update all features of an operational PDS, including additions and changes to the content, there must be a means of tracking and reacting to updates at runtime.

R13: Support for open systems. Creating high quality content is a significant problem for some display owners, for example the owner of a small shop. Therefore, providing access to a wide range of existing applications and sources of content would be highly beneficial to them [4, 29]. This would also be beneficial to researchers studying the impacts of existing or integrated applications on target users and user communities as it reduces the development effort. Therefore, a PDS platform should offer facilities for easily integrating existing applications and content services as envisioned by the kinds of open display networks that have been proposed [8].

Although the list of systems discussed in the previous section is by no means complete, it is apparent that, while all existing systems meet some of these requirements, none of them meet all of them. In the next section, we introduce a metamodel for PDSs that could provide the basis for developing a new generation of PDS platforms that cover all of these requirements and therefore could support a wide diversity of PDS applications, including those currently the subject of research.

PDS METAMODEL

In this section, we will present the most important parts of our metamodel for PDSs, discussing how they contribute to the list of general requirements. Fig. 1 shows the main concepts of the model and the relationships between them. At the heart of the model are concepts essential to PDSs such as *content*, *visualisation*, *curation*, and *context*.

Clearly, the *Content* concept supports **R1**, while including a separate concept for *Visualisation* supports the clear separation of content and visualisation **R2**.

The concept of a *Service* has been integrated into the model to support multi-service applications **R3**. Examples of information services would include BBC News, Twitter, weather information and events information. Each service may be associated with zero to many specific content types. For instance, a news service could be associated with *News Content* which is a subtype of *Content*.

To support heterogeneous displays **R4**, we introduced the concept of a *View* which is absent from most other PDS systems. Views are used to define how a *Visualisation* adapts to different display characteristics such as type, size and resolution. Accordingly, a *Visualisation* may be associated with one or more *View* objects, each of which is tailored to a specific set of display characteristics.

The model includes two additional concepts – *Service Curation* and *Context* – to support the customisation and context-dependent delivery of content and its visualisation. A *Service Curation* object includes filters and selectors that specify which content of a service should be delivered **R5**. *Context* refers to information that characterises a situation related to the interaction between users, applications and the surrounding environment. A *Context* object comprises rules that evaluate occurrences of a given context by means of different sensors. The introduction of a *Context* concept into the model allows the specification of what curated content and visualisation should be delivered by the system in a particular situation **R6**. The concepts of *Service Curation* and *Context* are very much related in that they tightly work together in the process of delivering the appropriate content and visualisation. Our context model is based on support of the requirement for pervasive sensing and interaction **R7**. Each *Context* includes one or more *Rules* that are evaluated over one or more sets of *Sensor Data* from specific sensors in the sensor network denoted by *Joined Sensor* which may be either a *Software Sensor* or *Hardware Sensor*. An example of a hardware sensor is a temperature sensor, while an example of a software sensor would be weather information extracted from an external website.

A *Display Configuration* object represents the configuration of an individual display and it can be dynamically updated at runtime **R8**.

To support dynamic display networks, we include the concept of *Joined Display* to represent displays which are connected to the system **R9**. Displays of any type may join the system and each of them will be subscribed to exactly one *Display Configuration*. A *Display Configuration* can be associated with any number of *Context* objects, each of which delivers exactly one *Visualisation*. A *Context* may require one or more *Curations* to be executed, where each *Curation* is associated with a specific *Service*. The black diamonds on *Service* means that instances of *Curation* and *Content* are deleted when their containing instance of *Service* is deleted, ensuring that there are no dependencies on services that no longer exist.

The user model used to address the requirement for user management **R10** is shown in Fig. 2. A user may be assigned many roles and each role is assigned a set of permissions.

Figure 2. User Model

Fig. 3 shows the workflow model we use to meet the requirement for workflow management **R11**. Reviewing the core model for *Content*, *Context*, *Curation*, *Visualisation* and *Display Configuration*, it is clear that each of these can involve various user roles throughout their life cycle. We therefore introduce a general concept *Element*. Each *Element* is associated with exactly one *Workflow State*. A state corresponds to a step in the life cycle and will determine whether or not an element is currently published.

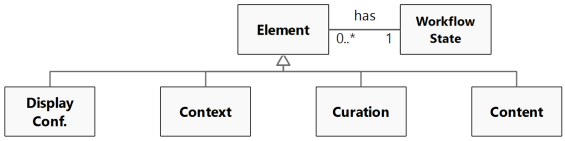

Figure 3. Workflow Model

Fig. 4 gives an overview of the update tracker part of our model **R12**. *Updated Element* refers to any object that is newly added or updated to the set of *Element* objects. When an *Updated Element* is detected by the update tracker, it will issue these updates to the associated set of *Joined Displays*.

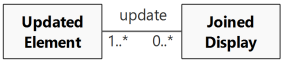

Figure 4. Update Tracker Model

The last part of our metamodel shown in Fig. 5 addresses the requirement to support open systems **R13**. This is achieved through a plugin integration model which consists of two components – external systems denoted by *Host Plugin* and the *Platform*. *Host plugin* refers to a repository where the plugins are made available. Thus, it may offer a number of plugins which can be integrated into the *Platform*. The model includes three types of plugins – *Service*, *Visualisation* and *Functionality*. *Service* and *Visualisation* refer to the concepts of *Service* and *Visualisation* mentioned earlier. An example of *Service* plugin would be a service that shows BBC news, while a Visualisation plugin can be a template which visualises the services in white and black colours. *Functionality* refers to functionality which extends and expands the system. An example of a *Functionality* plugin would be an extension that facilitates the management of multimedia.

SCREENPRESS

Based on the metamodel, we developed a prototype platform called ScreenPress that meets the proposed requirements. In this section, we will outline the main features of ScreenPress, using the example of an application to illustrate how the customisation and operation of a PDS is supported.

A layout and presentation of the application, called ambient news service, is shown in Fig. 6. On the left of the figure, there is the list of content feeds configured for this display which includes a list of services. One of the content feeds on the left is highlighted and its items are shown in the main area. The

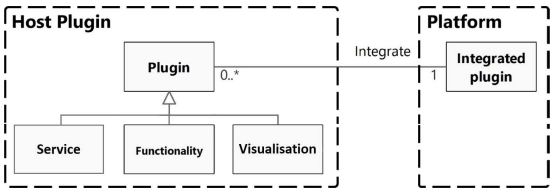

Figure 5. Plugin Integration Model

presented content may either be created by users, in which case it is stored and managed by WordPress, or come from external sources.

The application comes with a number of services commonly used in existing PDSs, allowing a typical installation to be easily configured without any implementation effort. At the same time, we wanted to include a range of services that would demonstrate the flexibility, and validate the architecture and technology choices of the ScreenPress platform.

The fact that different content can be shown on different devices allows the content to be tailored to specific contexts or user preferences. Devices may be added and configured dynamically through the WordPress administrative dashboard as shown in Fig. 7. For each display configuration (g), different contexts can be specified (a) by adding context rules (b) and content services (e) as well as selecting a template (d) for visualisation. Then, for each of the services, a panel enables users to specify filters (c). For example, we can specify a particular category of news, or a set of hashtags or users for Tweets.

Using ScreenPress, a PDS can be customised and extended by developers in several ways. WordPress defines websites using themes which can be activated or edited for the layout and presentation of a PDS as a whole, or for a particular content feed. Access to external content services is via plugins, and other services can easily be integrated by adding new plugins, e.g. from more than 45'000 freely available on wordpress.org.

ARCHITECTURE AND IMPLEMENTATION

ScreenPress is based on a client-server architecture and consists of three main components – WordPress, a distribution engine and an input preprocessing buffer – running on two servers as shown in Fig. 8.

The back server consists of WordPress running on an Nginx HTTP server. The core metamodel concepts of *Content*, *Curation*, *Context*, *Display Configuration* and *Service* were

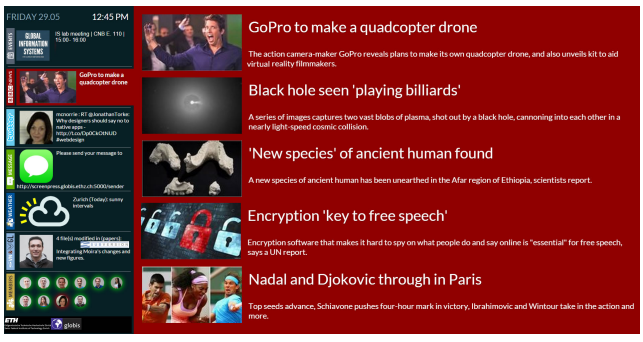

Figure 6. Configurable ambient news service app. screenshot

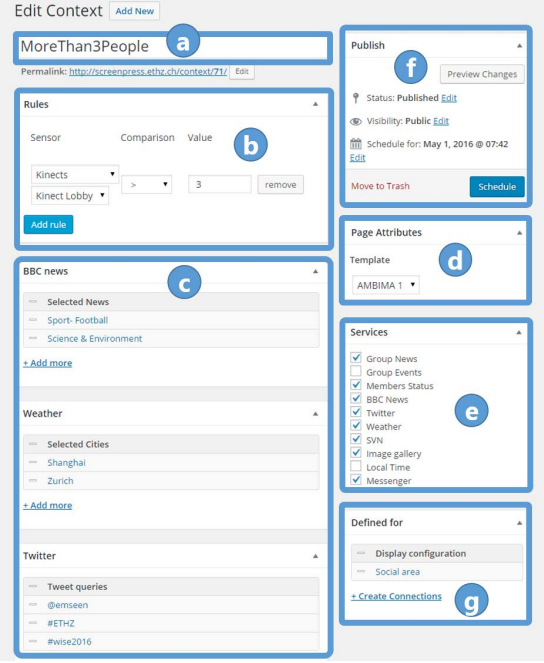

Figure 7. Ambient news service app. configuration – (a) context name; (b) context rules; (c) content curation; (d) template selection; (e) service selection; (f) workflow state; (g) associated displays configurations.

implemented within WordPress through the definition of custom post types and the Posts2Posts (P2P) plugin[2] to represent the associations between concepts. The concepts of *Visualisation* and *View* as well as the *User*, *Workflow*, and *Plugin Integration* models are provided and supported in WordPress by default and therefore required no implementation. To enable communication with WordPress, an RSS feed generates JSON format that provides curated content and the HTML of any templates.

The *front server* is split into two components—the distribution engine and the input preprocessing buffer. The server is implemented in JavaScript on top of Node.js[3] which was designed to build fast and scalable network applications.

The distribution engine implements the metamodel concept of *Joined Display*, providing advanced facilities for keeping track of available devices, and real-time publishing of content, visualisation, context, and interaction on different devices. While the WordPress component keeps track of updates related to system administration and locally created content, the distribution engine uses timers to check for new data of active external content services. Therefore both components work together to fully support the *Update Tracker Model*.

The input preprocessing buffer implements the metamodel concept of *Joined Sensor*. It is the core component responsible for collecting data from any sensors, preprocessing it and making it available to both the back server and front-end clients. The sensors can join via the *input preprocessing buffer* and their data are distributed via the *distribution engine*. When the state of preprocessed sensor information is changed, the

[2] https://wordpress.org/plugins/posts-to-posts/
[3] http://www.nodejs.org

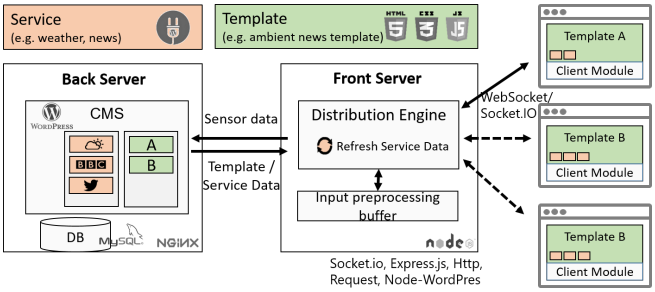

Figure 8. ScreenPress architecture

input preprocessing buffer sends the information to WordPress which evaluates the *Contexts* and, if there is a match, delivers the corresponding content and visualisation to the distribution engine which forwards them to related joined devices.

The client-side of ScreenPress is implemented using HTML5, CSS3 and JavaScript. The client-side module loads the template, dispatches messages from the front server and presents the received data to clients. Communication between the front server and clients is realised with Socket.IO[4] which provides bidirectional real-time communication.

A client can connect to the system using a web browser and a *session name*, and then receives a unique *session id* from the distribution engine. The distribution engine then sends an HTTP request to the WordPress component containing the device session name as well as current sensor data. WordPress matches the session name to the predefined configuration for the session name, checks the defined contexts and sends the curated data and UI for the matching context to the device.

Although we have developed different types of applications, our focus in this work is on one application. Table 1 provides an overview of the eleven services integrated into the ambient news service application with an indication of the underlying ScreenPress components. While a number of these services, such as BBC News and Twitter, are newsfeeds that purely involve the passive display of content and can be handled entirely by WordPress, others involve some form of user interaction and build on functionality offered by other ScreenPress components, namely the distribution engine and input preprocessing buffer as well as the front-end component.

Although many functionalities used in these application services mainly rely on WordPress, the incorporation of the other components into ScreenPress enabled us to meet all of our requirements for a PDS platform.

[4] http://socket.io

	Group News	Group Events	Members Status	BBC News	Twitter	Weather	Subversion	Messenger	Voice Navigator	Local Time	Voice Image Search
WordPress	•	•	•	•	•	•	•	•			•
Input Buffer Preproc.									•		•
Distribution Engine								•	•		•
Client module								•	•	•	

Table 1. Application integrated services with an indication of the ScreenPress components involved

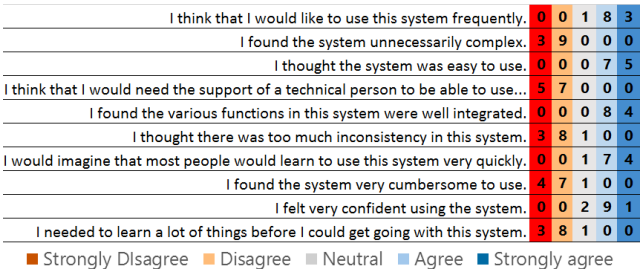

Figure 9. The results of SUS questionnaire.

USER STUDY

To evaluate the ScreenPress platform, we ran a user study in a controlled lab setting. Our experiment had two primary goals. First, we aimed to evaluate the usability of the platform. Second, we wanted to compare our platform with existing systems that are currently used for managing the content of various PDSs.

Participants

We recruited 12 participants (5 males; age: 18-59 (median=41)) from different faculties of our university who are responsible for managing the content of public displays. Their levels of education were varied (1 High school, 3 Bachelors, 5 Masters, and 3 PhDs). All of the participants had previous experience in managing the content of PDSs, in total 36 PDSs (median 2 per participant), and all had previous experience with CMS (5 with WordPress: 2 newbies, 1 intermediate, 2 advanced).

Methodology and Procedure

After introducing the system and the purpose of the study, we asked participants to perform different tasks for configuring two 65" LCD displays mounted in landscape mode. Each participant had to perform three tasks: (1) configure one of the displays based on our detailed configuration description; (2) configure the second display as they wanted; (3) add two content items for a content service – one had to include media and be published immediately, while the other had to be scheduled to be published in one minute. For each task, the participants were also asked to check whether the displays were updated according to their configuration and changes.

After the experiment, we asked participants to fill out a questionnaire comprising: demographic information, previous experience with CMS and questions from the Software Usability Scale (SUS) questionnaire [23]. The SUS consisted of 10 questions each with a 5 point Likert scale, resulting in a single measure of usability that is between 0 and 100. We used the SUS score as the main measure of the usability of ScreenPress. Above 68%, 74%, and 80.3% usability scores are considered as average (grade C), good (grade B), and excellent (grade A) usability performance, respectively [23, 24]. We performed one-sample Wilcoxon signed-rank tests (p = 0.05) to check whether usability performance of ScreenPress is statistically significantly different from these levels. In addition, the participants were asked to mention and give an overall rating to previous PDSs that they had used and give an overall rating to ScreenPress. To compare the overall rating of ScreenPress with other systems, we used a Wilcoxon signed-rank test (p = 0.05, two related samples). In the analysis, if a participant had used multiple previous systems, in order to be conservative, we considered the rating of the most highly-rated system.

Results

The detailed distribution of the responses to the SUS questions is presented in Fig. 9. The median system usability score of ScreenPress was 80.0% (IQR range: 73.1%-89.4%) which can be recognised as excellent according to the SUS guidelines [24]. One-sample Wilcoxon signed-rank tests showed that the usability score of ScreenPress is statistically significantly higher than 74% score (good level), $Z = 2.121, p = 0.034$.

The participants had already used a variety of platforms ranging from their own developed platforms to commercial products such as ScreenFOOD[5], InfoScreen[6], BrightAuthor [7], and FrontFace Public Display[8]. A Wilcoxon signed-rank test showed that the overall rating of ScreenPress was statistically significantly higher than previous systems used by the participants, $Z = 2.880$, $p = 0.004$. Indeed, the median overall rating for ScreenPress was 8.5 (IQR range: 8 to 9), whereas for previous systems it was 6.5 (IQR range: 5.25 to 7).

Discussion

The ScreenPress usability score (median 80%) is systematically higher than a good score and is at the level of the excellent usability grade [24]. This result indicates that we were able to retain the simplicity and familiarity of the WordPress administrative dashboard while extending it to support the configuration and management of a network of public displays.

In comparison to other systems, the interface of WordPress was found *"much nicer and more intuitive to work with"* (P6) and it was mentioned that *"a customized system such as screenpress would be quicker and easier to use"*(P4). Also, the fact that ScreenPress is a web-based system was considered *"very useful to manage the displays remotely out of office"* (P8).

The majority of participants expressed an interest in replacing their current PDS system with ScreenPress. Since they were all responsible for PDSs in a university environment, they asked for the integration of various university context related services.

CONCLUSION

We have presented a set of general requirements that a PDS platform should meet and used them to design a metamodel for PDSs. As proof of concept, we developed a PDS platform that is based on the model and implemented on top of WordPress. Since modern CMS, and particularly WordPress, support many of the core concepts and functionality required for PDSs, this seriously reduced the implementation effort and demonstrated the advantages of implementing a PDS on top of a CMS. Further, given the widespread use of WordPress, this approach also benefits from the fact that many content providers and PDS developers are already familiar with WordPress.

[5] www.screenfoodnet.com
[6] www.infoscreen.de
[7] www.brightsign.biz
[8] www.mirabyte.com

REFERENCES

1. Florian Alt, Stefan Schneegass, Michael Girgis, and Albrecht Schmidt. 2013. Cognitive Effects of Interactive Public Display Applications. In *Proc. 2nd ACM Intl. Symposium on Pervasive Displays (PerDis)*. ACM. DOI: http://dx.doi.org/10.1145/2491568.2491572
2. Jorge CS Cardoso and Rui José. 2009. A Framework for Context-Aware Adaptation in Public Displays. In *Proc. On the Move to Meaningful Internet Systems (OTM)*. Springer. DOI: http://dx.doi.org/10.1007/978-3-642-05290-3_21
3. Sarah Clinch, Nigel Davies, Adrian Friday, and Christos Efstratiou. 2011. Reflections on the Long-Term Use of an Experimental Digital Signage System. In *Proc. 13th Intl. Conf. on Ubiquitous computing (UbiComp)*. ACM. DOI: http://dx.doi.org/10.1145/2030112.2030132
4. Sarah Clinch, Nigel Davies, Thomas Kubitza, and Albrecht Schmidt. 2012. Designing Application Stores for Public Display Networks. In *Proc. Intl Symposium on Pervasive Displays (PerDis)*. ACM. DOI: http://dx.doi.org/10.1145/2307798.2307808
5. Sarah Clinch, Mateusz Mikusz, Miriam Greis, Nigel Davies, and Adrian Friday. 2014. Mercury: An Application Store for Open Display Networks. In *Proc. ACM Intl. Joint Conf. on Pervasive and Ubiquitous Computing (UbiComp)*. DOI: http://dx.doi.org/10.1145/2632048.2636080
6. Nigel Davies, Adrian Friday, Peter Newman, Sarah Rutlidge, and Oliver Storz. 2009. Using Bluetooth Device Names to support Interaction in Smart Environments. In *Proc. 7th Intl. Conf. on Mobile Systems, Applications, and Services (MobiSyS)*. ACM. DOI: http://dx.doi.org/10.1145/1555816.1555832
7. Nigel Davies, Marc Langheinrich, Sarah Clinch, Ivan Elhart, Adrian Friday, Thomas Kubitza, and Bholanathsingh Surajbali. 2014. Personalisation and Privacy in Future Pervasive Display Networks. In *Proc. 32bn ACM Conf. on Human Factors in Computing Systems (CHI)*. ACM. DOI: http://dx.doi.org/10.1145/2556288.2557287
8. Nigel Davies, Marc Langheinrich, Rui Jose, and Albrecht Schmidt. 2012. Open Display Networks: A Communications Medium for the 21st Century. *Computer* 45, 5 (2012). DOI: http://dx.doi.org/10.1109/MC.2012.114
9. Alexandre de Spindler, Stefania Leone, Matthias Geel, and Moira C. Norrie. 2010. *Using Tag Clouds to Promote Community Awareness in Research Environments*. Springer, 3–10. DOI: http://dx.doi.org/10.1007/978-3-642-16066-0_1
10. Corsin Decurtins, Moira C Norrie, Elke Reuss, and Nadir Weibel. 2008. AwareNews-A Context-Aware Peripheral News and Awareness Display. *Proc. 4th Intl. Conf. on Intelligent Environments* (2008).
11. Ivan Elhart and Nemanja Memarovic. 2013. WE-BAT: Web Based Application Template for Networked Public Display Applications that Show User Contributed Content. *Poster at 2nd Intl. Symposium on Pervasive Displays (PerDis)* (2013).
12. Luca Frosini, Marco Manca, and Fabio Paternò. 2013. A Framework for the Development of Distributed Interactive Applications. In *Proc. 5th ACM SIGCHI Symposium on Engineering Interactive Computing Systems (EICS)*. DOI: http://dx.doi.org/10.1145/2494603.2480328
13. Tommi Heikkinen, Petri Luojus, and Timo Ojala. 2014. UbiBroker: Event-Based Communication Architecture for Pervasive Display Networks. *Proc. IEEE International Conference on Pervasive Computing and Communication Workshops (PERCOM WORKSHOPS)* (2014). DOI: http://dx.doi.org/10.1109/PerComW.2014.6815259
14. Simo Hosio, Marko Jurmu, Hannu Kukka, Jukka Riekki, and Timo Ojala. 2010. Supporting Distributed Private and Public User Interfaces in Urban Environments. In *Proc. 11th Workshop on Mobile Computing Systems Applications (HotMobile)*. ACM. DOI: http://dx.doi.org/10.1145/1734583.1734590
15. Adriana Ispas, Sarah Schöni, and Moira C. Norrie. 2012. ARENO: Ambient Reminder Notes. In *Proc. 24th Australian Computer-Human Interaction Conference (OzCHI)*. DOI: http://dx.doi.org/10.1145/2414536.2414580
16. R. JosÃl', H. Pinto, B. Silva, and A. Melro. 2013. Pins and posters: Paradigms for content publication on situated displays. *IEEE Computer Graphics and Applications* 33, 2 (2013), 64–72. DOI: http://dx.doi.org/10.1109/MCG.2013.16
17. Tomas Lindén, Tommi Heikkinen, Vassilis Kostakos, Denzil Ferreira, and Timo Ojala. 2012. Towards Multi-Application Public Interactive Displays. In *Proc. Intl. Symposium on Pervasive Displays (PerDis)*. ACM. DOI:http://dx.doi.org/10.1145/2307798.2307807
18. Tomas Linden, Tommi Heikkinen, Timo Ojala, Hannu Kukka, and Marko Jurmu. 2010. Web-Based Framework for Spatiotemporal Screen Real Estate Management of Interactive Public Displays. In *Proc. 19th International Conference on World Wide Web (WWW)*. ACM. DOI: http://dx.doi.org/10.1145/1772690.1772901
19. Jörg Müller, Dennis Wilmsmann, Juliane Exeler, Markus Buzeck, Albrecht Schmidt, Tim Jay, and Antonio Krüger. 2009. Display blindness: The Effect of Expectations on Attention Towards Digital Signage. In *Proc. 7th Intl. Conf. on Pervasive Computing*. Springer, 1–8. DOI: http://dx.doi.org/10.1007/978-3-642-01516-8_1
20. Timo Ojala, Vassilis Kostakos, Hannu Kukka, Tommi Heikkinen, Tomas Linden, Marko Jurmu, Simo Hosio, Fabio Kruger, and Daniele Zanni. 2012. Multipurpose Interactive Public Displays in the Wild: Three Years Later. *Computer* 5 (2012), 42–49. DOI: http://dx.doi.org/10.1109/MC.2012.115

21. Steven C Rosenbaum. 2011. *Curation Nation: How to Win in a World Where Consumers are Creators*. Vol. 1. McGraw-Hill New York.

22. Amir E. Sarabadani Tafreshi, Kim Marbach, and Moira C. Norrie. 2017. Proximity-Based Adaptation of Web Content on Public Displays. In *International Conference on Web Engineering (ICWE)*. Springer.

23. Jeff Sauro. 2011a. *A Practical Guide to the System Usability Scale: Background, Benchmarks & Best Practices*. Measuring Usability LLC.

24. Jeff Sauro. 2011b. Measuring usability with the system usability scale (SUS). (2011). http://www.measuringusability.com/sus.php

25. Katarina Stanoevska-Slabeva, Vittoria Sacco, and Marco Giardina. 2012. Content Curation: A New Form of Gatewatching for Social Media?. In *13th Intl. Symposium on Online Journalism*.

26. Oliver Storz, Adrian Friday, Nigel Davies, Joe Finney, Corina Sas, and Jennifer G Sheridan. 2006. Public Ubiquitous Computing Systems: Lessons From the e-Campus Display Deployments. *IEEE Pervasive Computing* 5, 3 (2006). DOI: http://dx.doi.org/10.1109/MPRV.2006.56

27. Constantin Taivan and Rui José. 2011. An Application Framework for Open Application Development and Distribution in Pervasive Display Networks. In *Proc. On the Move to Meaningful Internet Systems (OTM)*. Springer, 21–25. DOI: http://dx.doi.org/10.1007/978-3-642-25126-9_4

28. Constantin Taivan and Rui José. 2014. Application Diversity in Open Display Networks. In *Proc. 3rd Intl. Symposium on Pervasive Displays (PerDis)*. ACM. DOI: http://dx.doi.org/10.1145/2611009.2611035

29. Constantin Taivan, Rui José, and Bruno Silva. 2015. Web-Based Applications for Open Display Networks: Developers' Perspective. *IJCSSE* 30, 1 (2015), 21–30.