Universal Access in the Information Society (2020) 19:409–432
https://doi.org/10.1007/s10209-019-00650-5

LONG PAPER

Concepts and design space for a better understanding of multi-device user interfaces

Fabio Paternò[1]

Published online: 4 April 2019

Abstract
This paper discusses the motivations behind and the characterising concepts of multi-device user interfaces by looking at the main design issues that have been addressed and the various solutions proposed. The discussion of relevant systems and frameworks highlights their main features, which are then used as the basis for comparative discussion. It compares different approaches and perspectives adopted in this area (e.g. responsive design, cross-device, distributed, migratory user interfaces). The features constitute a design space that can be used to facilitate analysis and comparison of tools and frameworks for multi-device user interfaces. Such aspects can be exploited by user interface designers and developers to analyse and compare various options when addressing existing and new applications. The analysis provided may inspire the design and development of new tools and frameworks as well.

Keywords Multi-device environments · Distributed and migratory user interfaces · Cross-device user interfaces

1 Introduction

One of the main issues in recent technological evolution is *device fragmentation*. Indeed, the mass market continuously proposes new devices with different features. For example, screen resolutions currently vary considerably amongst both smartphones and desktops: personal computers often have resolutions between 800×600 and 4096×2160 pixels, and smartphones can range between 320×240 and 2436×1125 or even 2960×1440 pixels. In the meantime, on the one hand, large public displays are becoming cheaper and cheaper, and pervasively installed in a variety of places (airports, shops, universities, stations, …) and on the other, smartwatches (e.g. Gear S 360×480) demonstrate that wearable computers can be exploited even independently of other devices. However, such a device fragmentation implies wide-ranging variety in terms of interaction resources and supports for media formats and apps, a situation which creates many interoperability and usability problems. A previous study [17] aims to identify the reasons why and how users interact with multiple devices in their daily life. The results indicate that people already exploit various techniques for cross-device information access and management. However, various issues still limit their use in terms of user experience. In particular, the study indicates that managing information across devices is the most problematic aspect when accessing multiple devices.

The technological evolution and related issues have stimulated fast growing interest in cross-device user interfaces. An early study [79] aimed to investigate the most important aspects for user experience (UX) when people access Web-based applications using various devices (usually desktop and mobile devices). In this perspective, they find that three main aspects should be addressed for improving cross-platform service UX: (1) how the application structure in the various devices involved supports actual user activities in order to achieve a satisfying task performance in the various possible settings, (2) the fluidity of the interaction flow across the devices so that it is perceived as well connected, and (3) coherent user access, which means that even if the users access it through different devices, the application is always perceived as a single unit.

However, managing information across multiple devices still requires further investigation. Indeed, some studies [33] found that we spend our online time with at least four interactive device types (smartphones, tablets, PC/laptops, TVs). It is possible to classify their use in: *Sequential*

✉ Fabio Paternò
fabio.paterno@isti.cnr.it

[1] HIIS Laboratory, CNR-ISTI, Pisa, Italy

usage, which means accessing interaction devices one after the other in order to complete a task, and *Simultaneous* usage, which means accessing multiple devices at the same time for supporting activities that can be either related or unrelated. Santosa and Widgor [72] report on interviews with 22 professionals working in various companies about their use of digital devices in order to identify new cross-device interaction patterns and possible problems. Their key findings indicate: the emergence of parallel use in addition to the traditional sequential use; cross-device interaction techniques to ease such parallel use are still lacking; a tendency to specialise the use of novel device types, indicating the need for design and development tools able to support such expected use; finally, data management is problematic because the increasing number of devices tends to fragment data through them. More recently, a diary study that involved fourteen participants [43] found the considered users actively have on average access to 7.9 information devices. For seven participants, some devices were provided by their employers. The smartphone was the only device that every participant said they use every day. In addition, they introduce a slightly different way to classify their usage: sequential use and parallel use, which is further divided into resource lending, related or unrelated parallel use, where resource lending means the participant's activity primarily focused on a single device, but this device borrows some resources from other devices. Common examples of resource lending included borrowing the input and output capabilities of another device. For example, the screen of another device could be used to display visual information or speakers to play audio. In related parallel use, participants work on a single task using two or more devices. All devices are involved in the same task. In most cases, the motivation for using multiple devices in parallel was to have multiple views of the task, dedicating different devices to different content or applications. A common situation is a participant using another device to view additional information when watching video content with a home media centre.

In general, the most problematic parts in designing multi-device user interfaces are: limited capability to adapt to the context of use; poor techniques supporting coordination in the performance of tasks carried out across various devices; and lack of effective support for carrying out activities seamlessly across multiple devices.

With the aim of helping user interface designers and developers to better address such aspects, the main objectives of this paper are:

- Introduce and discuss the main concepts that should be considered in order to understand the key issues to address when designing and developing multi-device interactive applications,
- Discuss the various possible ways in which the identified concepts can be addressed,
- Review the state of art in terms of techniques, languages, and tools in this area,
- Describe a design space for the possible solutions, which can be useful for analysing existing tools and frameworks as well as developing new ones in this area.

Particular attention is paid to issues and solutions in Web environments, which are the most commonly used, and the discussion also considers the use of relevant model-based techniques as well as how adaptation and continuity can be supported in multi-device user interfaces.

In particular, its main contribution is in proposing a set of concepts and a design space for a better understanding of multi-device user interfaces. Such results have been obtained from direct experience in designing and developing tools and frameworks for multi-device user interfaces for about fifteen years. We have, moreover, performed careful analysis of papers published on tools and support for multi-device user interfaces in human–computer interaction conferences and journals such as EICS, CHI, UIST, Mobile HCI, TOCHI, UAIS over the last twenty years and have examined previous proposals to provide design spaces. In this work, in addition to the impact of multi-device interfaces on users, we focus on architectural and functional aspects of the tools and frameworks that enable multi-device interactive systems. There have been previous contributions in this area [2, 15, 18, 56, 64, 71], but the set of dimensions that we propose aims to be more complete and concrete than the previous ones and has a number of dimensions that on the one hand enables making comparisons between different solutions in concrete terms and on the other is not too cumbersome or difficult to manage.

In the paper, there is an initial section on multi-device task patterns that aims to provide a high-level overview of the possible uses of multiple devices in supporting users' tasks and includes concrete examples to allow readers to better connect the introduced concepts with their direct experience. Next, then we indicate possible approaches in developing multi-device user interfaces (in order to introduce the developers' perspective), followed by a discussion of the context-dependent aspects that are becoming more and more relevant since such device ecosystems can be used in many different environments. Then, we move on to introduce a novel classification of multi-device environments that has implications both from the user's and the developer's perspectives (and provide examples for each of them in order to better highlight the differences), including a definition of cross-device user interface, which is generally overlooked in the literature. One type of such environments is characterised by the migration concept, which is then explained in detail with the various possible levels of support. The most

flexible type of environment is the one supported by cross-device user interfaces, and we then explain various important relevant aspects (distribution management, architectural support, involvement of wearables, target device representation, and selection). The eleven dimensions that characterise the design space are a consequence of such previous analysis and discussion and aim to provide a logical structure to facilitate comparison amongst approaches and identification of areas still underexplored. Indeed, such dimensions include distribution and migration, and the associated issues previously discussed such as device access, device selection, architecture, and other recurrent aspects addressed in some previous relevant tools and frameworks including granularity of the support, type of triggers, and modalities involved. A discussion of current research challenges concludes the article.

2 Multi-device task patterns

In terms of novel concepts for better understanding multi-device user interfaces, we also propose a set of multi-device task patterns. Such patterns indicate how the tasks are supported by the user interfaces of the platforms involved. Thus, they are defined by the specific relationships that exist between user interfaces, tasks, and platforms. The term platform in this case refers to sets of devices with similar interaction resources (for example, the smartphone, the smartwatch, the desktop, …). In particular, we can identify the following patterns:

- *Same task, same user interfaces* The same task is supported through the same user interface on the various platforms considered. This is the case when whatever device is used the user interface to support the considered task is almost the same; for example, a login is substantially supported in the same manner over different platforms.
- *Same task, different user interfaces* The same task is supported on the various platforms considered but through user interfaces that differ in some presentation or layout aspect, as often happens with applications developed with the responsive design approach.
- *Same main task, different subtasks* The same main task is supported on the various platforms considered, but with some different related subtasks. This refers to when the main task can still be performed on the various platforms, but it has different associated secondary tasks depending on the platform considered. For example, a hotel booking application, which on the smartphone requires filling in a limited number of fields in order to make the task efficient, on the desktop provides the possibility to enter more detail (further secondary tasks), under the assumption that the user has more time and availability to provide them.
- *Different, but related, platform-dependent tasks* Dependencies exist amongst tasks (and corresponding user interfaces) carried out on different platforms. This means that what is done on one device has an effect on what can be done on another. Thus, some tasks performed on the various platforms may be different but are related to each other. Various types of such dependencies can be identified, for example *enabling* (when one task accomplished on one platform enables some task on another platform).
- *Platform-specific task* These are tasks that are relevant only on specific platforms. This is the case when the tasks depend on specific features available only on certain platforms, for example because they require a lot of screen space or are associated with some mobile positions or specific peripherals (e.g. a camera or a step counter).

These task patterns provide a high-level overview of the possible uses of multiple devices in supporting users' tasks. For each of them, it is possible to identify more concrete design patterns. For example, some possible design patterns for the case in which there are some dependencies amongst tasks carried out on different platforms are reported in [57].

In the following, we can see some examples of such task patterns. Figure 1 illustrates a case in which the same task is supported through user interfaces that differ on the platforms considered. Indeed, we can see that both user interfaces support the possibility of booking a flight, but they use different interaction techniques. The left version is oriented to a desktop system and is optimised for mouse-based interaction through the use of pull-down menus and radio buttons, while the right version is designed for a touch-based mobile device with large buttons and interaction techniques facilitating selection by finger.

In the example in Fig. 2, the two versions have the same main tasks (related to filling in the fields To, Subject, cc, …), but the mobile version does not provide immediate support for some secondary tasks (e.g. save draft, insert attachment, modify text properties).

The next example (Fig. 3) refers to the case when there are dependencies between tasks carried out on different platforms. The example shows an application that enables the user to control (start or stop) a video displayed on a TV through a smartphone. Thus, by interacting through one device (in this case the smartphone), it is possible to determine what is presented on the other one (the TV).

The last example (Fig. 4) in this section shows a case of tasks meaningful only on a specific platform. It is about an activity tracking application in which on the desktop, it is possible to visualise spatial analytics describing the movements performed during the day, while on the mobile, it is

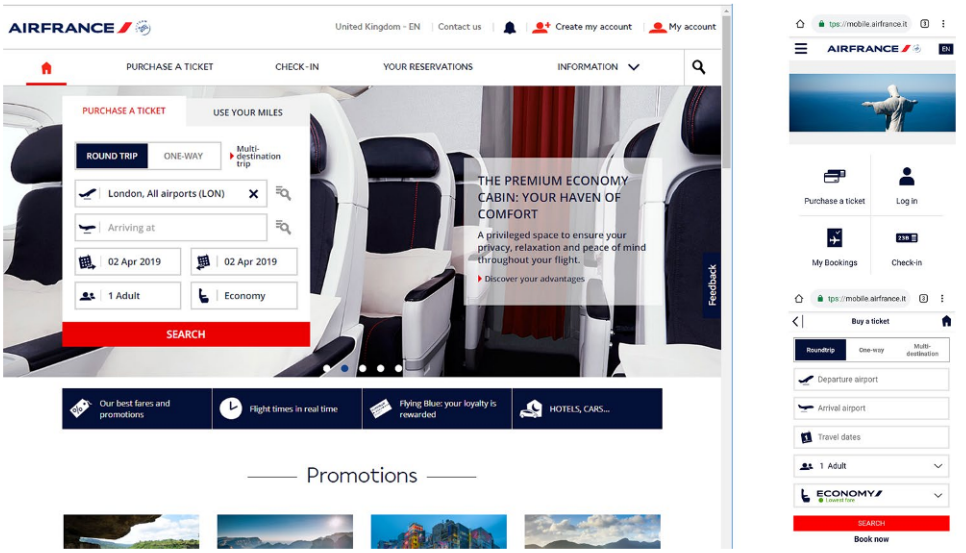

Fig. 1 Example of task supported through different user interfaces in different platforms

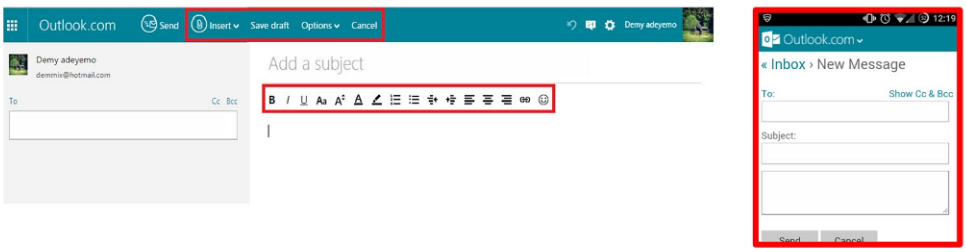

Fig. 2 Example of task with different secondary tasks in different platforms

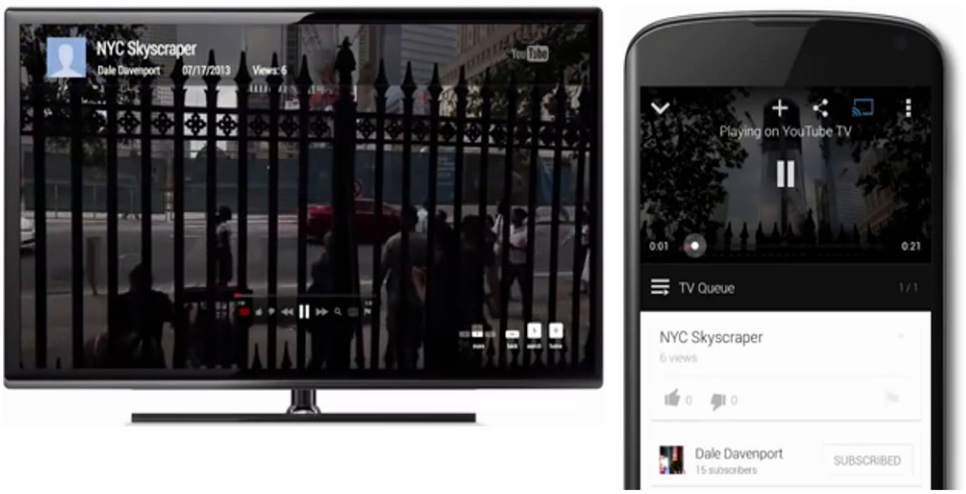

Fig. 3 Dependencies between tasks carried out on different platforms

possible to get real-time data regarding a running exercise with indications of distance travelled, calories consumed, and average pace.

3 Authoring multi-device user interfaces

There are various possible approaches to authoring multi-device user interfaces:

- *Platform-specific authoring*, which has the advantage of allowing better control over final results, but requires more effort in development and maintenance, since the effort for the most part has to be replicated for the number of platforms addressed;
- *Model-based authoring* [8], which uses user interface conceptual descriptions in order to avoid dealing with many low-level implementation details for the various

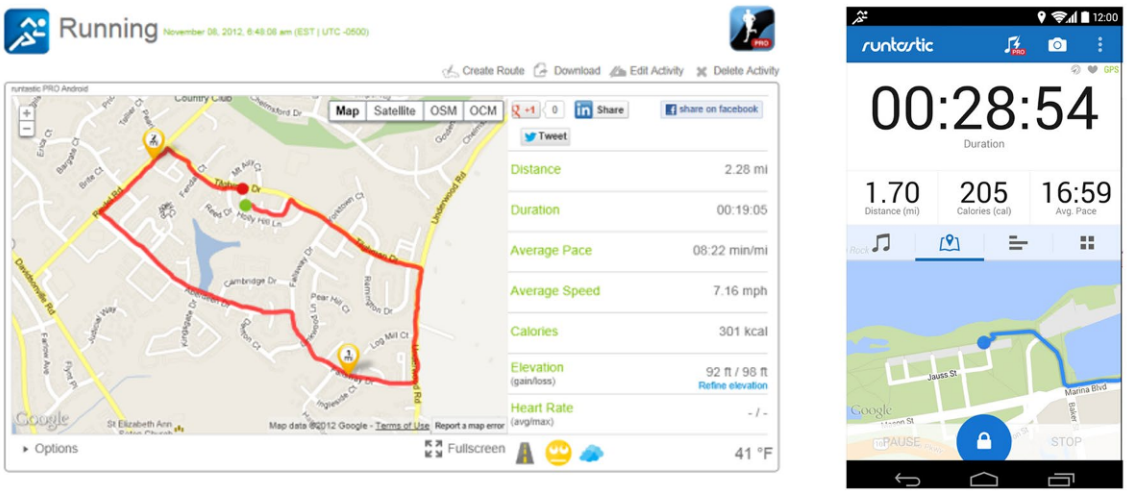

Fig. 4 Tasks relevant only on specific platforms

devices addressed, and user interface generators to render adapted user interfaces;

- *Responsive web design* [48] (also supported by tools and frameworks such as Webflow, Pingendo, Bootstrap), which utilises one main version with fluid layout and subversions, though the possibilities are limited to media query support and graphical interfaces;
- *Distributed authoring* [59], in this case it is possible to edit and show the user interfaces distributed across the various possible platforms at the same time.

Model-based approaches for multi-device user interfaces exploit languages for describing the main aspects in conceptual terms. Such languages can describe interactive systems at various abstraction levels: in terms of tasks and domain objects, or abstract descriptions of the user interface (which means using a vocabulary independent of the interaction modalities), or concrete descriptions (in this case the description takes into account the interaction modality but is independent of the coding language). One example of authoring environment in this area is Supple [24], which is an intelligent tool able to take as input the indication of the supported tasks, the constraints specific to the platform considered, a relevant example scenario, and a cost function reflecting user preferences, and finds the design able to optimise the cost function still satisfying the constraints associated with the considered platform. It has also been applied to creating accessible applications, for example with interfaces optimised for users with impaired dexterity. Another example of environment adopting the model-based approach is MARIAE (the MARIA Environment) [64], which supports editing and analysis at the various possible abstraction levels, and then user interface generation for different modalities and implementation languages. In general, such model-based approaches enable designers and developers to concentrate on the main conceptual aspects and avoid using a multitude of coding languages by linking semantic information and implementation elements. Thus, they facilitate device interoperability through many possible implementation languages, and the support of assistive technology. HTML5 has adopted some model-based concepts by introducing a number of tags that characterise more explicitly the semantics of the corresponding elements, but it is limited to obtaining graphical interfaces, while some model-based languages, such as MARIA, also support multimodality (for example combining graphical and vocal interactions) through the definition of specific concrete user interfaces. GUMMY [54] retargets UIs from one platform to another by adapting and combining features of the original UI. For this purpose, it builds a logical model in the background, generating initial designs which can be refined at a later stage. Data-driven approaches have been explored [44] to automatically transform desktop-optimised pages for other devices. They use existing pages on the target platform as a source for examples, and then produce pages that both address the technical limitations of the platform and are based on what users actually do on these devices.

Currently, one popular approach to addressing the variety of available devices is responsive design, which is particularly oriented to Web applications. In practice, through the use of media queries, it is possible to detect

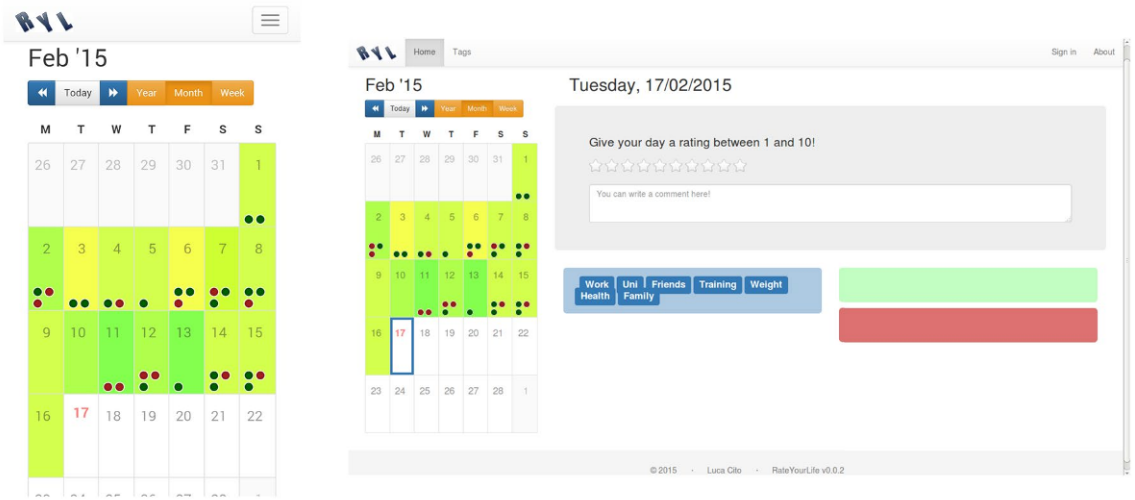

Fig. 5 Example of responsive design application

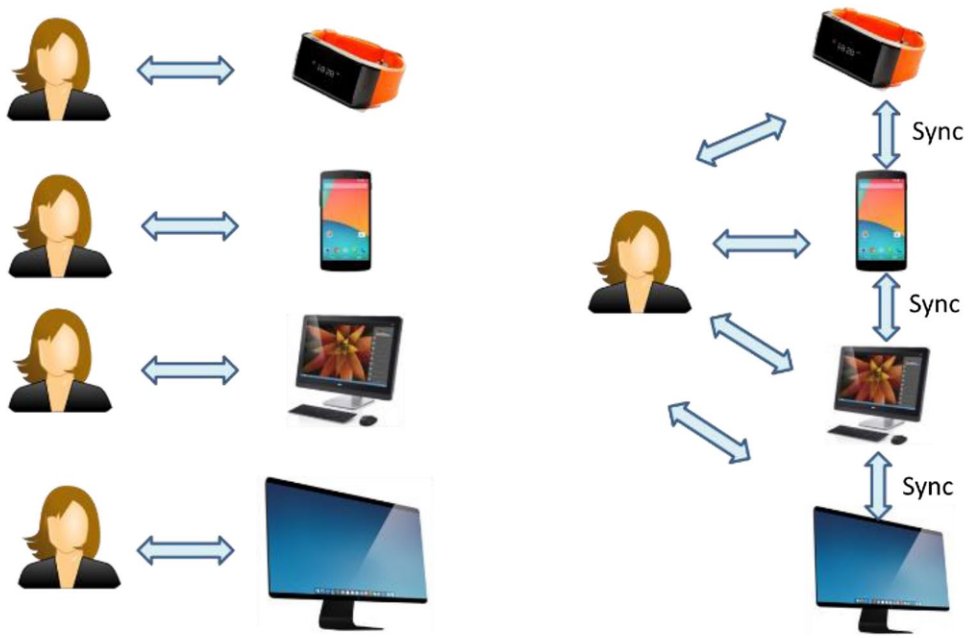

Fig. 6 From responsive (left) to cross-device (right) user interfaces

a small number of device aspects, such as screen width, height, orientation, aspect ratio, colour, resolution, etc. It is mainly used to identify groups of devices with similar resources (usually screen size). One example framework supporting such approach is Bootstrap,[1] which identifies four breakpoints to classify the target devices: "extra small devices—phones (< 768 px); small devices—tablets (≥ 768 px); medium devices—desktops (≥ 992 px); large devices—desktops (≥ 1200 px)". An example application obtained through this approach is shown in Fig. 5. It is an application that allows users to quickly rate and comment their daily experience. The left side illustrates the layout shown on a device 640 pixel wide, which is an overview of a month showing the colour-coded personal state for each day, while the right shows the view on a ≥ 992 pixel wide device also showing the details of a specific day with personal comments in addition to the overview.

However, the basic assumption of responsive design is that each user at a given time interacts with only one device; such device may vary over time, and thus, it provides the possibility of easily modifying the presentation mainly according to the screen size. In cross-device design, such assumption no longer holds. As Fig. 6 shows, in cross-device

[1] http://getbootstrap.com/.

interfaces, the user can interact with multiple devices at a given time, and the various parts of the user interface are distributed across the different devices and keep their state synchronised.

Multi-masher [38] aimed to apply the mash-up approach to multi-device environments. The goal was to support development of multi-device web applications based on the reuse of existing web sites created for single device usage. The limitation in this case is that it is not possible to create new components for the cross-device user interfaces. One example of a distributed authoring environment for obtaining such cross-device user interfaces is XDStudio [59], which aims to support two modes that can be considered when designing distributed UIs. The simulated mode enables designing for a multi-device environment while working on only one device, which simulates the other involved devices. The on-device mode enables distributing the design process across multiple devices at the same time, since design and development are conducted directly on the considered devices.

A first exploration of how to support testing of such cross-device user interfaces is reported in [58]. Cross-device sessions are represented as data containers, which record any data exchanges across devices, and allow session playback in order to facilitate data sharing between sessions. This is obtained through two tools. The session controller enables managing and grouping different devices into different sessions. Devices can be either real or simulated. The session inspector provides for scrutiny and analysis of the sessions. It provides support in order to record/replay cross-device sessions and monitor any data exchange during such sessions.

4 Context-dependent adaptation

When designing user interfaces accessed through multiple devices, developers also have to consider the possible contexts of use and adapt the user interface to it in order to support users more effectively. Context has been defined as any information that can be used to characterise the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application [19]. We can identify four main dimensions that characterise more concretely contextual aspects:

- *User*, which involves aspects related to the task, mental state, physical activities, preferences, knowledge, etc.;
- *Environment*, which relates to aspects concerning the actual physical environment and its attributes such as noise, light, and temperature;
- *Devices* and their features such as screen size, connectivity, multimedia support, and orientation;
- *Social aspects* that are able to identify various possible relations (friendship, belonging to groups, etc.), which may be relevant to deciding what to show in the user interface.

Any element in such dimensions can have an impact on how to change the design in order to improve the user experience. User interfaces are composed of various aspects that can adapt dynamically, including presentation, which consists of how the elements are arranged and the associated attributes; dynamic behaviour, which indicates how it is possible to navigate across the available elements; and content, which provides the information communicated.

For each of such aspects, there are various specific adaptation techniques. For example, in adaptive navigations, there are techniques such as direct guidance, or dynamic hiding/showing links, or links annotations [6]. One potential useful content adaptation technique in multi-device user interfaces is page summarisation, whose aim is to automatically reduce content. Thus, it can be used to obtain an amount of content that fits well in the available screen space. There are two main approaches in this area: abstraction-based, which exploits techniques oriented to manipulating texts, such as reduction, compression, and reformulation; extraction-based, which rates sentences with the aim of identifying those best representing the entire content, and for this purpose exploits feature-based (e.g. attributes, sentence position, term frequency, …), machine learning, and graph-based techniques. An example of the extraction-based approach is PowerBrowser [7], in which a word within a passage is ranked as important when it is contained there frequently, but infrequently in a larger, more general, collection of documents. The sentences that contain the highest number of different words that occur frequently in close proximity are thus considered as the most important.

Another interesting opportunity in adaptation is provided by crowdsourcing tools. Such tools enable users to directly personalise the arrangement of their user interfaces, for example when the application is accessed through specific devices. This makes it possible to discuss and share adaptations with others who access the application in the same context (e.g. the same device and similar preferences). An example application of this type of approach is described in [60].

In general, context-dependent adaptation rules may be structured through three concepts: events, conditions, and actions. This type of structure has been used in the area of active database management systems [14]. In this case, it is used for modelling different dynamic behaviours where:

- An event is something that happens at a given time and occurs interacting with the application or in the surrounding context of use.

- A condition is a specific constraint on the state of some contextual aspect that must be satisfied for longer periods of time. Events and conditions can be combined with logical or temporal operators. Temporal operators can indicate a requested sequencing of events or that events should occur at specific times.
- The action defines the desired effect. It can be the activation of some functionality or a desired change in the user interface.

When the effect is a change in the user interface, it can have an impact at different granularities by involving: the entire UI, some UI parts, single UI elements, or only attributes of single UI elements. In some cases, the change is not limited to some specific attributes of one element or a group of elements, but it involves the relations between some UI elements, for example, moving an element before or after another one. Examples of adaptation rules structured in terms of events, conditions, and actions are:

- Event (background noise rises to over 35 dB); condition (the user is slightly hearing-impaired); action: all videos have to display subtitles.
- Event (the battery charge changes level); condition (if the battery charge is less than a certain level); action (reduces screen brightness)
- Event (the user runs a given application); condition (the user is over 70); action (increase all font sizes by 10%)
- Event (it is dinner time); condition (user is outdoors); action (display nearby restaurants and their locations)
- Event (user starts walking fast); condition (shopping list app active); action (wider spacing for shopping list items).

Some concepts related to such adaptation rules have been adopted by IFTTT[2] (IF This Then That), an environment that allows users to connect various Web applications (such as Facebook, Evernote, Meteo, and Dropbox) through rules called applets. The part associated with the *This* indicates the trigger, while the *That* indicates the action. Trigger examples are: "when I'm tagged in a picture on Facebook" or "if tomorrow's weather forecast is for rain". Action examples are: « send text message" o « create a message on Facebook". IFTTT supports simple rules (combinations of only one trigger and one action). Unfortunately, IFTTT does not really support adaptation of interactive applications or multi-device user interfaces, but simply activations of some predefined functionalities. At research level, some proposals (e.g. [31]) have been put forward to extend this approach to actions that implement user interface modification and distribution across different devices. Thus, users can create rules such as: "When I am close to the TV duplicate the smartphone user interface on it".

One contextual aspect that is particularly relevant in cross-device user interfaces is the spatial relations amongst the involved entities. For this purpose, the proxemics interaction approach [49] has been proposed. It is based on considering a set of spatial relations between people, objects, and devices: orientation, identified by the relative angles between the considered entities (e.g. devices) in the environment; distance, the distance between entities in the environment (for example the distance of a user from a public screen); motion, dynamic movements (such as a user moving towards a display to access the information that it shows); identity, information regarding the various entities; and location, which allows the definition of the setup in the environment considered, in which some entities have fixed locations (for examples walls and doors), while others are movable. One issue is that knowledge of proxemics may be easily exploited to the detriment of the user. Some authors [34] review a set of dark patterns in which proxemics interaction can be misused, also discussing the underlying problems and possible solutions that could increase their appropriate use. An example of such dark patterns is when a user enters a specific area to carry out an extended activity that does not require system support, and the system detects the presence and offers unsolicited (and potentially undesired) support.

Indeed, while context-dependent adaptation can be a powerful tool to obtain flexible solutions, it is important that users can understand how such solutions work and control them. Bellotti and Edwards [4] discuss the need for intelligibility and accountability: context-aware systems that seek to act upon what they infer about the context must be able to represent to their users what they know, how they know it, and what they are doing about it. In addition, context-aware systems must enforce user accountability when, based on their inferences about the context, they seek to mediate user actions that impact others. Indeed, the incorporation of context awareness raises a number of issues. For example, users are required to trust the behaviour of the system's intelligence, and this requires the system to exhibit predictable behaviour and the ability to successfully and consistently preempt the user's goal. Unfortunately, intelligent applications may sometimes incorrectly preempt the user's goal, owing to either flawed intelligence or to incorrect contextual information [12]. In such circumstances, the user is likely to feel frustrated, because the application will either appear overly prescriptive or, worse still, present incorrect adaptations.

[2] https://ifttt.com/.

5 Possible types of multi-device environments

In this section, we discuss different technological solutions for supporting multi-device user interfaces and provide a classification of the various possible levels of support, with examples of what users could do with each of them in order to show their potential impact.

5.1 Evolution of multi-device user interfaces

Multi-device interaction has been investigated for several years. One of the first interaction techniques supporting distributed user interfaces was pick-and-drop [69], which allowed users to select some data on one device user interface and drop it on another device through pen-based interactions. Thus, it allowed pieces of information or files to be easily moved across devices, but it did not support migratory user interface, since it was not possible to move interactive components from one device to another. ConnectTables [76] still supported pen-based interactions across devices, but with the additional possibility to detect when two displays are close to each other so that they can be considered a unified homogeneous display area. The issue of using displays projected on tables as a spatially continuous extension of the user's portable device was addressed in augmented surfaces [70], which supported the possibility to drag data from the portable computer to the table. However, such solution does not support mobile users, since it requires the use of a fixed video camera above the table.

Multibrowsing [40] was one of the first examples of support for moving Web pages across various displays. It provides support for coordination control within a set of Web browsers on distinct displays. This was an interesting approach, but the enhanced clients need a custom browser plug-in, and the target displays have to run a special service to receive the distribution events and open the pages. In addition, there is only one available command (browse(URL)), which has to be applied to the entire user interface. iRos [41, 66] is the corresponding middleware platform, which was introduced as a meta-operating system aiming to connect devices that have their own low-level operating system. iRos exploits Event Heap, a framework based on events that uses a shared tuple-space, thereby enabling services to register to events in "a distributed blackboard fashion". It also provides a Data Heap abstraction, which supports a mode of operation designed for data sharing, and iCrafter [65], a service framework implemented by exploiting Event Heap. Similar features have more recently been supported by Substance and Shared Substance [32], an approach that aims to be simpler and more coherent by using a tight connection involving data and events. For example, iRos requires design-time decisions regarding accessing elements through events, data, or both. On the other hand, this aspect can be managed at run-time through Shared Substance. Overall, iRos supports moving Web pages from screen to screen, but it is only possible to move a complete user interface and not parts of it, and there can only be two types of distribution: one-to-one and one-to-many. Moreover, the movement of Web pages across devices does not maintain the UI state.

Multi-device support is emerging also in widely used environments and applications. For example, the YouTube app allows users to render videos on nearby TVs. Chrome sync allows showing the list of the tabs across the users' devices so that users can access any of them when they want. Likewise, Firefox provides a tool[3] that synchronises bookmarks, passwords, and history across desktop and mobile devices. Pushbullet[4] is another tool that allows notification mirroring, and the sending of links, files, messages, and images between devices through the use of browser extensions and mobile apps. Research efforts have also been put forward to better manage multi-device environments. For example, ActivitySpace [37] aims to consider activity-centric resource management across various types of mobile devices by exploiting "an interactive desk as mediating configuration space". Such space displays the devices and resources taking part in the activity to be supported. Further devices may be included by indicating their position in the relevant configuration space. Resources can be simply dragged and dropped to a device.

5.2 Classification of multi-device user interfaces

In general, in the current technological settings characterised by a wide availability of various types of devices, it is frequent to have some level of multi-device access to application functionality. However, there are various possibilities for actually supporting multi-device access. Thus, in order to clarify the distinction between such possibilities, we introduce the following classification:

- *Sequential application access through different devices* (by using a single device at any given time), as happens with applications obtained through responsive design tools;
- *Distributed user interfaces* more than one device can be used to enter inputs to the application functionality;
- *Migratory user interfaces* in which it is possible to dynamically move user interface components across devices while maintaining their state;

[3] http://mozilla.com/mobile/sync.

[4] https://www.pushbullet.com/.

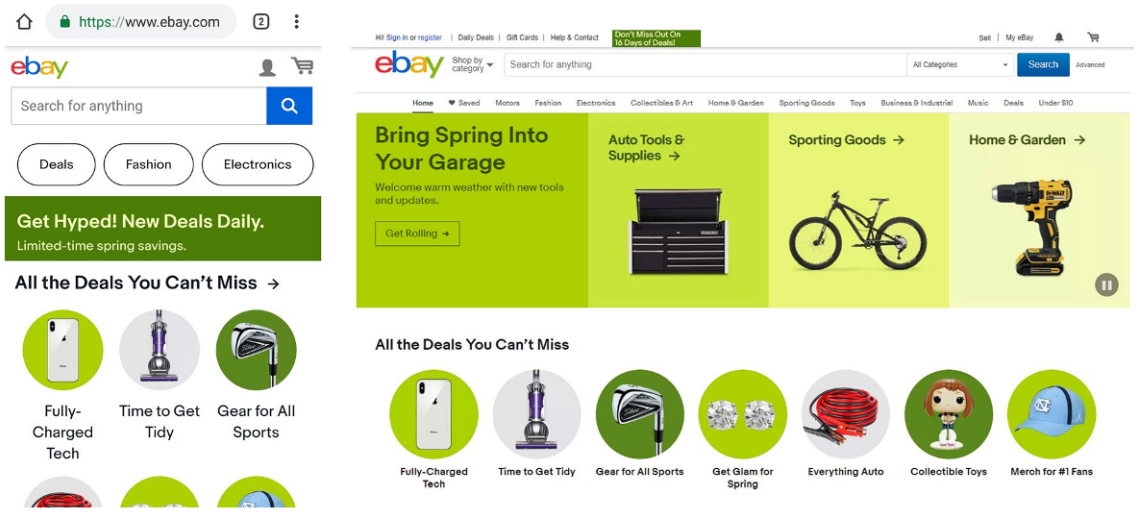

Fig. 7 Example of sequential application access through different devices

Fig. 8 Example of distributed user interface

- *Cross-device user interfaces* which refer to user interfaces whose components are distributed and keep their state synchronised whatever device is used for interaction. They allow redirecting input events from one device to another and support high-level events obtained by combining events generated in different devices.

In this classification, both distributed and cross-device interfaces refer to cases in which users access the application through multiple devices at the same time. The difference is that in the case of cross-device user interfaces, there is also a synchronisation obtained through event redirection or composition across the devices at the user interface level. Such aspects are not supported in basic distributed user interfaces, in which the shared state is only that of the application server-side part (as happens with some cloud-based applications). Thus, they are able to show updates in the various parts of the user interface (for example updates to a connected database, which is a server-side component), but not the state changes in the user interface components that are distributed in different devices.

In order to clarify the differences between the four approaches listed above, we can consider an example application (an online shopping list). The sequential approach typically refers to a responsive Web site that allows users to interactively select the items that should be part of their shopping list through one device. The way to present and select such items depends on the device currently in use, smartphone or tablets or personal computer (see example in Fig. 7).

An implementation of such service able to support distributed user interfaces would have to allow users to interact with the shopping list through multiple devices at the same time. So, for example, the user should be able to use at the same time both her smartphone and a display located in the shop (see Fig. 8), the smartphone is used to edit parameters for searching amongst the available products database, and

Fig. 9 Example of migratory user interface

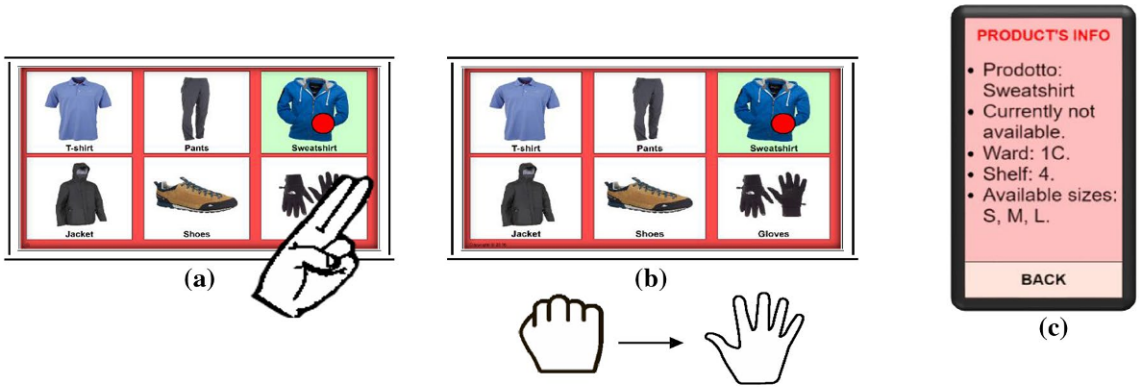

Fig. 10 Example of cross-device user interface

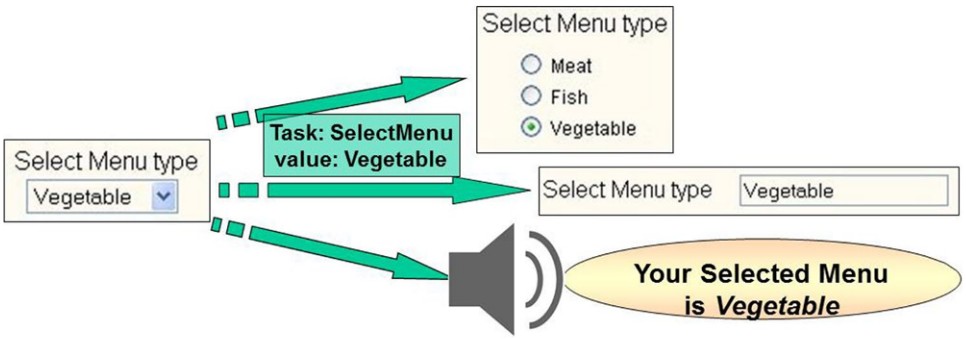

Fig. 11 Example of migration across devices preserving state

the public display is used to return the query results in a more readable format.

An implementation supporting the migratory features should instead offer the possibility to dynamically move parts of the user interface from one device to another. For example, a user would be able to activate the migration of the user interface part for specifying the parameters for querying the currently available offers from a kiosk in the store and move it to her smartphone in order to interact with it more easily (see example in Fig. 9).

In a cross-device implementation the user would be able to edit the shopping list by interacting at the same time with multiple devices with synchronised user interfaces components able to communicate and compose events from different devices. For example, the user can interact with a personal shopping list in the smartphone and a public display in the shop, so that when an item is selected in the personal list then the public display shows relevant offers so that the user can browse and select items through mid-air gestures, and the details of the selected element are immediately shown on the personal device (see example in Fig. 10).

It is important to note that such categories are partially overlapping, but there are also user interfaces that belong exclusively to one of these categories. For example, there are user interfaces that are at the same time distributed and migratory; this means that they are able to support access

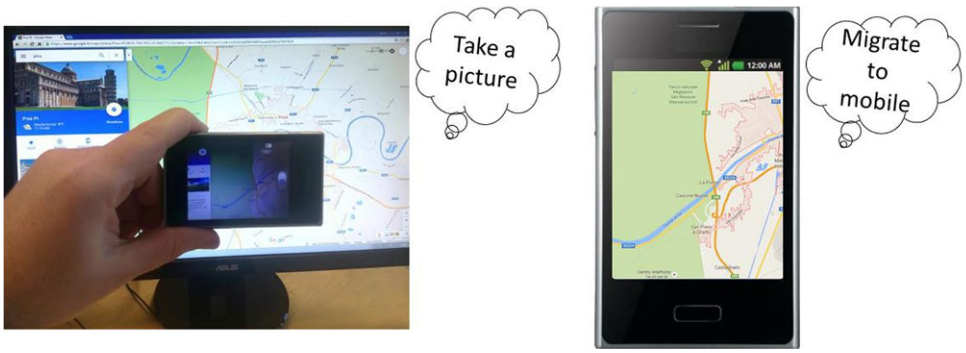

Fig. 12 The process supported in deep shot

through multiple devices and at some point some parts of the user interface can be moved across devices while preserving their state, but they may not be cross-device according to the above definition, because they are not able to keep the user interface components synchronised when they are accessed from different devices simultaneously. In the next two sections, we discuss some contributions in the areas of migratory and cross-device user interfaces.

5.3 Migratory user interfaces

The main motivation for migratory user interfaces has been the observation that one source of frustration in modern technological settings is that often users must restart the applications every time they change their device. Therefore, seamless access to interactive services across various devices is needed. Migratory user interfaces are able to move between "source" and "target" devices, and preserve their state (an example is in Fig. 11), in order to enable users to seamlessly accomplish their tasks across them. A number of approaches (e.g. pixel replication, browser session migration, virtual machines) can be adopted for this purpose. Application domains that can benefit from migration are those which imply long sessions with users on the move, such as shopping, online auctions, games, making reservations.

The state of the user interface that should be preserved after migration involves the results of all the possible user actions (e.g. tap or click, scroll, editing), including the element in focus, and the entire application environment, which, in the case of Web applications, means cookies, sessions, history, bookmarks, and JavaScript. For example, Myngle [75] provided support to facilitate users when changing devices during Web navigation through browser extensions and a native application. It facilitated visits to previously accessed pages through an integrated Web history functionality able to record interactions with the various personal devices, and filter such histories according to high-level categories.

An environment supporting migration has been Deep Shot [9], which is a framework and run-time support based on a picture taken on the mobile phone. The tool detects the screen capture area (see Fig. 12), the corresponding foremost application, and some information on the application state. Then, it transmits this information to the smartphone encoded in a URI, so that it can be activated on the smartphone with the same state. Using Deep Shot calls for installing some software on all the devices involved in the migration. In its processing, Deep Shot exploits existing computer vision techniques to determine which application is displayed on the screen, and then the desktop software extracts and communicates the corresponding URI.

A different approach [28] is based on a migration proxy server and migration clients, which are Web applications that should be open in the devices potentially involved in the migration. In this case, after the initial device discovery phase, users can freely access any Web application with the support of a migration proxy server, which includes in such applications some scripts that are then used to obtain the migration. Through the migration client users can determine the target device for migration and trigger it. When the migration is triggered, the injected scripts send the DOM and the state of the current page to the migration server, which creates a version of the considered page for the target device and uploads it so that the users can continue their activities from where they left off. In this solution it is also possible to support partial migration. In this case, not all the user interface is migrated, but only a subset of its elements that is dynamically identified by the users through the scripts that are included by the migration server. In [29] it is illustrated how this approach can be exploited in order to make customisation persistent. This means that once the partial migration is performed, then the user can navigate on the new device through user interfaces that are consistent with the user's previous selection (when accessing similar pages). Thus, for example, if the user accesses eBay and only the upper search bar and the first 6 items on its results page are migrated, then on the target device, when a new search is performed through the upper bar, the resulting page only presents the upper search bar with the first 6 results.

There are some cases in which, in order to make the migration process more fluid, it would be convenient to

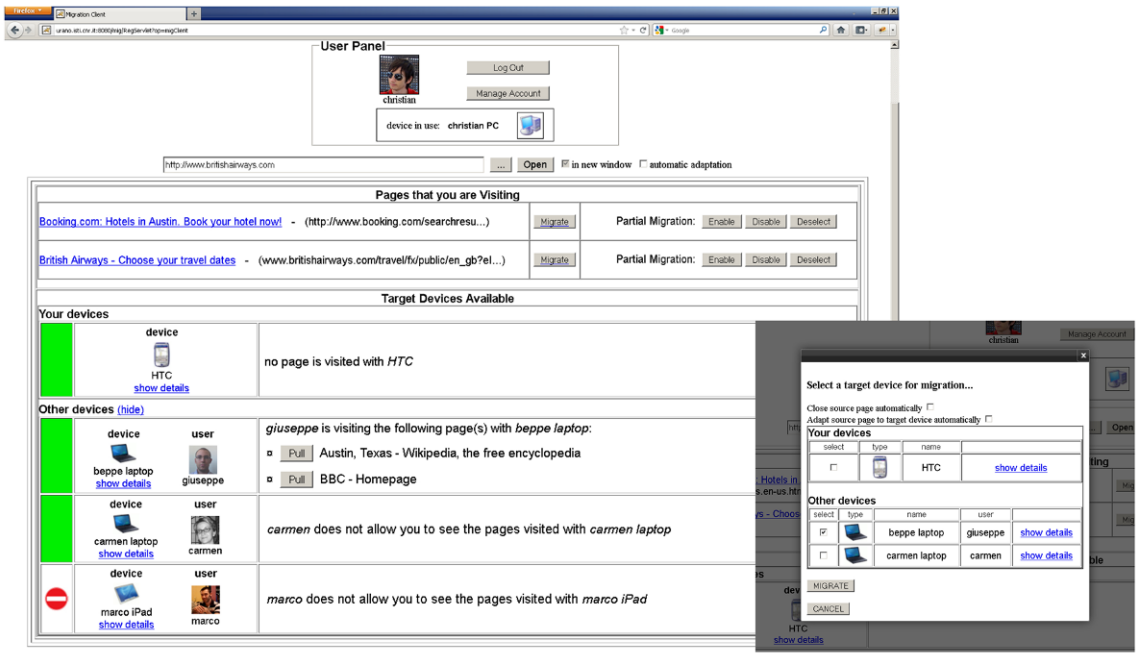

Fig. 13 Migration client supporting both push and pull

allow automatic target acquisition and selection, and suggested or automatic migration triggers [30]. This is useful when there are dynamic situations that occur frequently. For example, it would be possible to trigger an automatic migration from the smartphone to the office desktop when the smartphone is on a table, facing upwards and close to the desktop (the user has arrived in his office) and likewise, the inverse migration (desktop to smartphone) when the mobile device has been picked-up from the table, is moving, and the desktop is no longer close (the user is leaving the office).

One problematic issue in preserving the state of Web applications during migration is Javascript. Indeed, if the JavaScript variables' states are not saved and restored properly, incongruences may result when the user migrates. A first attempt to address this issue through the inclusion of additional JavaScript to save, transmit, and restore the state of the original application scripts was presented in [5]. A proposal for further extending such approach in order to better handle function closures, event-handler state, and HTML 5 rich-media objects was put forward in [46].

More generally, when the migration environment is able to support both push and pull of user interface components, it opens up various interesting collaborative ways to interact amongst users in different places and with different devices. An example scenario is the case when two colleagues have to organise a business trip. The first user starts to browse various Web sites in order to find good flight and hotel reservations. At some point, she finds a good deal and fills in the reservation form with all details concerning dates, room, personal contact references, and payment. Before submitting the form, she migrates (through a push operation) the corresponding page to the colleague. The second colleague can make his hotel reservation without having to browse the Web site and enter various details (only the personal contact references will need to be updated) and then decides to see what the first colleague is looking at for the flight reservation. Thus, it activates a migration (in pull mode) of the airline page currently accessed by her. Then, he also accesses the weather forecast web site but then remembers that he has to attend an urgent meeting; thus, he partially migrates only the relevant parts of the weather Web site to his smartphone in order to look at them in detail while moving to the meeting room.

Figure 13 shows an example of a migration client supporting both push and pull [28]. For each user it indicates the currently navigated applications with the possibility of dynamically migrating them to other devices. If the Migrate button is pressed, then a dialogue appears to specify the possible target devices for the migration. In addition, the migration client also shows the other devices involved in the migration environment, and, when the current privacy policies allow, it is also possible to see the applications that the corresponding users are currently accessing with the possibility of pulling them from such devices to one's own device.

Such environments can raise various privacy issues that need to be addressed through specific policies instantiated by the users involved. Such policies have to consider various aspects for each device: visibility, which users or devices

should be able to detect the device presence; activity, which users or devices should be informed of the applications currently accessed in the device; and migratability, which users or devices can target the device for migration.

In terms of usability, there are some specific aspects that are relevant for migration. Since continuity in terms of task performance across different devices is one of the goals of migratory user interfaces, the possibility of performing the migration in a time efficient manner becomes important. In this case, the time required from the trigger occurrence on the source device to UI perception on the target device is usually considered. In addition, the transition should be understandable by users, thus the resulting adaptation on the target device should not make it confusing for the user to continue in the interaction. Under this perspective, another important principle is predictability, which concerns what UI parts will migrate and in which part of the target device, and where user interaction results are going to be shown after the migration has been completed.

Panel Type	Physical Size	Keyboard Quality	Touch Quality	Proximity to User
Video Panel	●●●●●			
Playback Panel			●●●●●	●●●●●
Search Panel	●●●	●●●	●●●●	●●●
Related Videos	●●●		●●●●	●●●●

Fig. 14 How Panelrama [80] classifies the various user interface parts

5.4 Cross-device user interfaces

As already mentioned, distributed user interfaces support application access through multiple devices at the same time. Dynamically obtaining user interfaces with elements distributed across different devices helps users collaborate (e.g. to perform collaborative searches), improve group experiences (e.g. in guided tours), perform a task on the most suitable device, or use one device in coordination with others (such as using a mobile device to control a public wide screen). In order to be considered as cross-device, user interfaces should be able to synchronise the various parts of the distributed user interfaces and support redirection and composition of their input events. Thus, cross-device user interfaces are distributed user interfaces with additional features. An example of framework supporting their development is CDI [3]. It is a JavaScript library that provides an API to detect single and combined gestures, as well as management of cross-device interactions. It provides the possibility of defining combined events that associate events occurred in different devices and determine the transfer of state information across the involved devices.

Distribution management More generally, how to distribute user interfaces across different devices can be specified at various times:

- Design time, when the interactive application is specified, by indicating how the various parts should be allocated to the involved devices;
- *Design-time definition + run-time execution* in this case some dynamic changes in terms of distribution are specified through event handlers indicated at design time that are triggered by specific events at run-time;
- *Run-time* in this case distribution is obtained through dynamic customisation tools, which enable end users to determine distribution configurations not considered at design time.

Various proposals have been put forward that apply different strategies in supporting distribution authoring. Dygimes [78] was an environment for model-based UI development. The authors indicate that it is possible to distinguish static and dynamic approaches for UI distribution. Models are used to develop specifications at design time, and the executable UIs are obtained at runtime by exploiting such models. One more recent approach is Panelrama [80], which categorises device features in order to dynamically change the allocation of the various panels to those devices that best correspond to the desired features. For this purpose, it allows developers to specify how suitable the various panels are to different types of devices. Figure 14 shows an example indicating how from a usability perspective screen size is more important in a video stream panel with respect to a playback control panel, while for the playback panel touch quality and proximity to the user are much more important. This approach is based on an optimisation algorithm (similar to that used in Supple) in order to distribute panels to devices in such a way to obtain the best match for the designer's intent. In addition, as devices can be dynamically connected or disconnected to the distribution environment, the panels are dynamically reallocated to the devices available according to such optimisation algorithm.

A model-based approach to support distribution is introduced in [53]. It is based on a concrete user interface model for distributed user interfaces and a set of primitives

to specify the desired distributions in the generated user interfaces. Another model-based solution [47] is based on an extension of the MARIA language [63] and allows the specification of the distribution at various granularity levels (user interface, single presentations, composition of elements, single elements, parts of single elements) by applying the complementarity, assignment, redundancy, and equivalence (CARE) properties (originally introduced for multimodal user interfaces in [16]). In addition, it allows users to dynamically change the distribution of the elements to better satisfy unplanned situations by directly selecting the relevant parts of the user interface and changing the corresponding CARE property.

Solutions that do not require fixed servers One issue in cross-device user interfaces is that in some cases the access to a fixed server for their management is not possible. One possible solution has been put forward in the MU-DUI (Multi-Users Distributed User Interfaces) framework for cross-device interaction [23]. It was one of the first contributions that addressed support to multi-users roles in a cross-device user interface framework. It supports the possibility to specify the target device for the UI elements in various ways: specific device(s) identifier(s), device group(s) identifier, specific user(s), and user groups associated with specific roles. It dynamically detects the involved set of devices. One of the devices also holds the distribution engine component, which receives the distribution requests and maintains the current distribution state. The engine component can migrate on request from one device to another. It does not use a fixed server to support the distribution, thus it can manage offline mode: ad hoc devices network with no internet (one of the devices acts as access point among them), and therefore without communicating with the external world. Since it can work even without a fixed server, it can support cross-device user interfaces in a broader set of scenarios. A different solution that does not require fixed servers is investigated in Connichiwa [73], which runs local Web applications to achieve network independence. A native helper application automatically runs a Web server on demand on one of the joined devices. Other devices can then access the Web server through a shared network (such as an existing Wi-Fi network) or an ad hoc network. The purpose is to eliminate the need for a remote server, keep communication local and communication delay to a minimum. More peer-to-peer support was investigated in [52]; however, this approach used a specific extension of the Tcl/Tk development toolkit, which limited its applicability.

Cross-device user interfaces involving smartwatches More recently, the smartwatch has emerged as another interesting device type to be involved in possible user interface distributions. For example, WatchConnect [36] aims to support the rapid development of cross-device interaction techniques and applications based on smartwatches. It is based on a specific modular hardware platform able to emulate a smartwatch, with support for user interface components that can be exploited for facilitating the development of cross-device interactions through a set of predefined event handlers and simulators. Duet [10] aims to explore more generally a design space concerning possible interactions involving smartphones and smartwatches. Taking into account their spatial configurations, the proposed solution is able to coordinate the touch-based interactions with the two devices, in such a way as to extend their capabilities. As a result, the smartwatch enhances various smartphone-based interactive tasks. For example, the interface between the phone and the watch can be divided using the smartphone to show a canvas while using the smartwatch to host a palette. The environment supports cross-device novel gestures; an example is pinch-to-close that allows users to mute the two devices at the same time. It is also possible to exploit the smartwatch sensors to obtain specific gestures, for example, by detecting the part of the finger that is actually interacting with the smartphone during a touch. If different types of devices are considered (e.g. public display), then the Duet design space must be revised, because it depends on the features of the involved devices. More explicit support for developers of cross-device interfaces involving wearables can be provided by Weave [13]. This is obtained through a script language supporting some abstractions for selecting target devices, performing output actions on selected devices, and handling input events on one device. One limitation in Weave is that each device involved in the cross-device interaction must have installed a native app (Weave proxy), which provides run-time support useful to execute the relevant services.

Target devices representation and selection One important issue concerns how to provide users with awareness of the available devices. Fisher et al. [22] discuss the RELATE support and how it represents the devices near the freely moving users. Their experiment compared three interfaces, a spatial representation, a map with icons, and a textual alphabetic list for discovering and selecting devices within a certain proximity. The results showed users' preference for the representation proving a map with icons. The conclusions indicated that additional visualisation techniques would be needed when a high number of devices are available to preserve usability. Differently from the RELATE approach, which focused on spontaneous meetings, other works [27] concentrated more on users on the move seeking to select some devices available in multi-device environments. Therefore, they focused on representations of the surroundings and the available devices rather than their relative positions. For this purpose, two techniques were analysed: map-centred, in which the map of the surroundings was fixed in terms of position and orientation, and the user representation in it changes depending on how she moves about; and user-centred, in which the user icon is in the centre and never

changes its position or orientation, while in this case user movement is reflected by the map translating and rotating about her. More recently, the use of head-worn displays to facilitate the creation of a model able to identify the spatial relations amongst the surrounding devices has been investigated [74], but the technology used has still some limitations that make it not yet comfortable for mobile users. Having knowledge of devices spatial information can be useful for example to more easily select the target devices in cross-device interaction, rather than using menu lists by device ids that are often difficult to map with the devices available in the current environment. Indeed, a study [68] implemented three cross-device interaction techniques (one without and two with some spatial representation) and assessed them in a laboratory user test. The technique without spatial representations considered was similar to that proposed in Conductor [35], which uses menus with colour-coded device names to select the tablets to share information with or to chain tasks across them. For the spatially-aware interaction techniques they used radar views and edge bubbles, with the support of the HuddleLamp vision tracking [67]. Their results indicate that techniques with some spatial representation are more usable, if carefully designed to support direct interaction and avoid excessive mental effort in understanding the current position of the user and the devices. More recently, [42] investigated how to easily move graphical objects across personal mobile devices by comparing three techniques. The results seem to indicate that there is no specific method that is generally most usable, but the preferred technique depends on various aspects, such as the number of objects to move, the number and characteristics of the devices involved, the social situation and its implications in terms of privacy and security. In the same area, [20] have studied the mid-air gestures needed to complete data-transfer activities also with remote devices through armband-fitted hands. Results have shown that users are influenced by the spatial relationship between the devices involved in the interaction, in particular for transferring elements from one device to another, since this factor influences the direction in which the gesture is performed. However, there is a lack of studies to investigate which gestures support more intuitively the use of personal devices and public displays in combination, for example to accomplish tasks such as visualising extra information about content visible in the public display on mobile and wearable devices.

6 Design space for multi-device user interfaces

In order to summarise and focus our discussion and compare some of the contributions in this area, it can be useful to explicitly indicate some logical dimensions that characterise multi-device user interfaces. In this section, after discussing some previous proposals in the area of logical frameworks for multi-device user interfaces, we present a design space that is based on the concepts and the possible approaches discussed in the previous sections (user interface distribution, migration, adaptation, and architectural support). The logical framework can be used to describe the main features of existing solutions, evaluate how they are supported, and generate suggestions for features that are missing but could be relevant.

In previous work, Coutaz et al. [15] introduce an ontology for multi-surface interaction, which has a broad scope since it considers various types of user interface systems, and thus presents abstract concepts to describe them (such as topology, coupling and compatibility amongst surfaces) without addressing any particular architectural aspects. An abstract formal approach to specifying coupling of interaction resources and reasoning about its properties is introduced in [2]. Another contribution [56] focuses on a specific task (moving an object from a tablet to a tabletop display) and provides a corresponding design framework. In our case, we consider distribution of user interfaces of interactive applications; thus, we also address the distribution of interactive components (not only data objects) and its implications, and our analysis is not dependent on a specific type of task. An abstract reference model [18] has been proposed for analysing distributed interfaces according to four dimensions: computation (which parts are distributed), communication (when the UI is distributed), coordination (who it is distributed to), and configuration (from which device to which device the distribution occurs). However, these dimensions do not seem sufficient to conduct a complete analysis. Another study [71] considers user interface granularity, interaction space, and support for state distribution, which are relevant dimensions but not sufficient for a complete analysis of the various proposals put forward in this area.

We aim to propose a design space which is more complete and concrete than the previous one; hence, we focus on frameworks and tools that provide direct support for multi-device applications. For this purpose, our literature analysis and review have also been supported by our direct experience in the design and development of several tools and applications for multi-device user interfaces. The proposal is based on the discussion reported in previous sections, and revises and extends the dimensions introduced in [62] by providing a broader set of aspects to consider (such as how devices are selected or the support for multiple roles), clarifying the meaning of the proposed dimensions and better defining their possible values. Such dimensions highlight the main concrete aspects that can characterise systems and tools for supporting multi-device user interfaces, and a set of possible values for each dimension are used to indicate how each tool addresses

Table 1 A comparison amongst a set of representative approaches

Tool	Dimension										
	Distribution	Migration	Granularity	Trigger	User device access	Device selection	Modalities	Generat.	Adaptat.	Architect.	Multi-roles
Pick-and-drop [69]	Static	Data	UI elements	On demand	One-to-many	Device id	Monomod.	Predefined	None	Static networks	No
Multibrowsing [40]	Dynamic	Not supported	User interface	On demand	One-to-one	Device name	Monomod.	Predefined	None	Client/server	No
Dygimes [78]	Dynamic	Not supported	User interface/ groups	On demand	One-to-many	Device id	Monomod.	Runtime	Transduc.	Client/server	No
Peer-to-peer DUIS [52]	Dynamic	UI elements	User interface/ groups/UI elements	On demand	One-to-many	Device name	Monomod.	Runtime	Transduc.	Peer-peer	No
Deep Shot [9]	Not supported	UI elements	User interface	On demand	One-to-one	Device id	Monomod.	Predefined	Scaling	Static networks	No
Proximity toolkit [49]	Dynamic	UI elements	User interface/ groups/UI elements	Automat.	Many-to-many	Device name, distance	Graphical and mid-air gestures.	Predefined	Scaling	Client/server	No
Web migration [28]	Dynamic synchronous	UI forms/ in part Javascript functions	User interface/ groups/UI elements	On demand (push and pull)/Automat.	One-to-one	Device name	Graphical and vocal	Run-time	Transduc.	Client/server	No
Duet [10]	Static	Data	UI elements	On demand	One-to-many	Device id (smartph./ watch)	Monomod.	Predefined	None	Static networks	No
MU-DUI framewok [23]	Dynamic synchronous	UI elements	User interface/ groups/UI elements/ parts of UI elements	On demand	Many-to-many	Device name, type, role	Monomod.	Predefined	Scaling	Without fixed server	Yes
Conductor [35]	Dynamic	UI elements	User interface/ groups/single elements	On demand	One-to-many	Broadcast/ device id	Monomod.	Runtime	None	Client/server	No
Multi-masher [38]	Dynamic synchronous	UI elements	User interface/ groups	On demand	One-to-many	Device id	Monomod.	Predefined	Transduc.	Client/server	No
Huddle lamp [67]	Dynamic in limited space (table)	Data	UI elements	On demand	One-to-many	Camera-based	Monomod.	Predefined	Scaling	Client/server	No
Panelrama [80]	Dynamic	UI elements	Groups	Automat.	One-to-many	Device type	Monomod.	Predefined	Scaling	Client/server	No
Weave [13]	Dynamic synchronous	UI elements	UI elements	On demand	One-to-many	Device name, type	Monomod.	Predefined	Scaling	Client/proxy/ server	No
Connichiwa [73]	Dynamic synchronous	UI elements	User interface/ groups	On demand	Many-to-many	Device id	Monomod.	Predefined	Transduc.	Peer-to-peer	No

Table 1 (continued)

Tool	Dimension									
	Distribution	Migration	Granularity	Trigger	User device access	Device selection	Modalities	Generat.	Adaptat.	Architect.	Multi-roles
Improv [11]	Dynamic synchronous	UI elements	User interface elements	On demand	One-to-many	Device id	Monomod.	Predefined	None	Client/server	No
Distr. MARIA [47]	Dynamic synchronous	UI elements	User interface/groups/UI elements	On demand	One-to-many	Device id	Monomod.	Predefined	Transduc.	Client/server	No
XDBrowser2.0 [57]	Dynamic synchronous	UI elements	User interface/groups/UI elements	On demand	One-to-many	Device name, type	Monomod.	Predefined	Transduc.	Client/server	No
CDI [3]	Dynamic synchronous	UI elements	User interface/groups/UI elements	On demand	Many-to-many	Device type/gesture	Graphical and mid-air gestures.	Predefined	Transduc.	Client/server	No
ADAM [61]	Dynamic synchronous	UI elements	User interface/groups/UI elements	Automat.	Many-to-many	Device type	Monomod.	Runtime	Transduc.	Client/server	Yes

it. In general, we consider devices with the ability to support both input and output. We identify a number of dimensions that would be sufficient to carry out comparisons between different solutions in concrete terms but, on the other hand, we want to avoid an excessive number of dimensions that would have made the design space cumbersome and difficult to manage and apply. The result is a design space composed of a set of 11 logical dimensions. Their application to a group of representative proposals can be reported in a table (such as Table 1), which is a possible way to exploit the proposed design space by making it possible to identify combinations of values in different dimensions that have not been particularly well investigated.

6.1 Distribution

Distribution indicates *whether* user interface parts on multiple devices are supported at the same time, and whether they are *dynamic* (when the user interface elements can vary their allocation to the devices during a user session) or *static* (when the distribution configuration cannot change during a session). We also indicate when *synchronisation* of the various distributed user interface parts is supported (an important feature to obtain fully cross-device user interfaces).

We can see that in general dynamic distribution is supported by the considered tools, with the exceptions of the tools focused on moving user interface components from one device to another to support mobile users (Web Migration proxy-based platform and DeepShot). HuddleLamp provides support for dynamic distribution, but only across devices that are on a table equipped with an overhead depth camera.

6.2 Migration

Migration specifies whether device change and state preservation are supported and which state aspects are preserved; such aspects can concern: *data*; *interactive forms* (which means all the interactive parts of the user interface); *user interface elements* (in this case only the state of some types of user interface elements is preserved during migration); *functionality*; *sessions*; etc.

Regarding migration, various tools support it at the level of user interface elements and preserve their state when they move from one device to another. The Web migration platform [28] also supports preserving the state of the session, and parts of the Javascript functionalities. Some approaches (e.g. pick-and-drop) are limited to transferring data but not user interface components.

6.3 Granularity

Granularity describes the levels of user interface detail addressed, which can be the entire user interface, groups of elements, single elements, and even parts of single elements.

In terms of granularities addressed to support the proposed functionalities, in general there is a good range of possibilities (entire user interface, groups of elements, single elements). In a few cases (e.g. [23, 47]), even parts of single elements can be addressed. Panelrama focuses only on panels, which are groups of elements that are associated with a specific intent, and Deep Shot considers only migration of the entire current user interface.

6.4 Trigger

The type of trigger activation indicates how the support is activated, it can be *on demand* from the user (which can be via *push* or *pull* or *both*) or *automatic* or *mixed* (which means that automatically some suggestions are provided and then the user can accept or modify them).

While the Web migration platform was able to support both user-generated triggers (either in push or pull mode) and some automatic triggers based on contextual events, the others mainly consider either automatic triggers, as in Panelrama and Proximity toolkit (which support triggering based on the detection of some type of device in the proximity in order to activate the distribution of some user interface parts), or triggers based on user requests. One recent contribution to support automatic distribution for multi-role user interfaces is ADAM [61], which extends some features of Panelrama concerning support for automatic distribution. In particular, while Panelrama considers only single users interacting with multiple devices, ADAM targets collaborative multi-user applications.

6.5 User device access

User device access indicates possible relationships between users and devices in the multi-device environment considered. Such aspects are also discussed in [77], by also taking into account the scale of the considered ecosystem, which depends on the size of the largest device involved. There are various cases: *one-to-many*, when the support is mainly for one user interacting with multiple devices; *one-to-one*, which means that each user interacts with only one device, even if the effects of their interactions can modify the user interfaces of other devices, for example, by distributing some component to them; *many-to-one*, which indicates that multiple users can interact with one device at the same time, i.e. a device shared by multiple users (e.g. a tabletop or a large screen); *many-to-many*, where multiple users can interact with public shared devices as well as with their personal device. The one-to-many case is supported by Improv [11], a tool that allows end users to augment the user interface of an existing application on one device (e.g. a laptop) using the input elements of an additional mobile device. The many-to-many feature is supported for example by the Proximity toolkit.

6.6 Device selection

One important aspect is how to select the target device for a distribution or migration. Some environments just provide the *name* of the available devices, but users may not understand what actual devices correspond to the identifiers listed. Other approaches (e.g. [27]) provide the possibility of selecting devices from a *graphical representation* of the surrounding environment with the indication of where the devices are located in it, but this requires preparation of the graphical representation in advance. CDI supports the possibility of identifying target devices through gestures that can be immediate to perform. In other cases, the possibility of connecting devices through *physical touch* has been explored [42], which is intuitive even if it may not be generalisable in environments with various types of devices ranging from large displays to wearables. In Panelrama, this aspect is addressed automatically based on the device type. (It aims to identify the best matching between features of the nearby devices and the types of information contained in the panels.) Another possibility is to identify a set of devices through the *role* of their users. For example, in a cross-device mobile city tour application, it is possible to identify at least two roles (the tourist and the guide) and decide that some interactive elements be presented on the devices of only one of them. This is usually managed through distribution profiles (see for example [23]). Another aspect is supported by the Proximity toolkit [49], which is able to identify the position and *distance* of the persons and devices in the environment, and thus supports selection of devices whose position satisfy some distance constraint. Some approaches limit the possible types of devices that can be selected (for example, Duet only supports smartphones and smartwatches).

6.7 Modalities

Interaction modalities indicate whether the user interfaces supported are *monomodal, transmodal* (only one modality at a given time but which can change) or *multimodal* (multiple modalities at a given time).

Regarding modalities, various approaches have addressed mainly graphical user interfaces, often in Web environments. The Proximity toolkit is integrated in Microsoft Visual Studio and has two plugins for the marker-based Vicon motion capturing system and the Kinect sensor, the latter of which allows tracking of skeletal bodies. CDI supports cross-device

Web user interfaces with the possibility to consider also mid-air gestures recognised through Kinect devices. The Web migration platform provides some adaptive multimodal support that is achieved through a model-based language and the corresponding generators for multimodal interfaces combining graphics and voice [26].

6.8 Generation

It indicates the type of UI activated, that is, whether the parts that are processed dynamically are *predefined*, *runtime generated* or *mixed*. With *mixed,* we refer to the case in which the UIs can be partially predefined and partially dynamically generated. The migration platform and the peer-to-peer toolkit are examples of solutions that support some level of dynamic generation of user interfaces, while the others exploit use of pre-developed components. In the case of the migration platform, this is obtained through the support of a model-based language, and it is more useful when a change of interaction modality is requested because of a change of context of use. Indeed, run-time generation supports more radical changes in the user interface, but it can also be time-consuming (because the user interface generation can take time depending on the user interface complexity), and thus it can limit usability if it occurs often because users can perceive the delays due to such generation. In addition, the results of run-time generation may not be optimal, and sometimes difficult to predict exactly [55].

6.9 Adaptation

It specifies the adaptation approach, whether it is performed by *scaling* the elements, *transducing* (elements can be changed but the overall user interface structure remains the same, as it happens in responsive design), or *transforming* the entire user interface structure by changing its layout and, in some cases, also the types of widgets.

Some level of adaptation that takes into account the targeted devices is provided by some tools (e.g. XDBrowser, the migration platform and the peer-to-peer toolkit). Complete reorganisation of the user interface according to the device can be too risky due to possible disorientation effects on the user, and the tools considered in the table do not support it.

6.10 Architecture

Architecture can be server-based, without fixed server or peer-to-peer. In terms of software architecture, in general, the client/server approach is supported, also because most of the available tools are oriented to Web applications, which have native support for it. The MU-DUI framework [23] is able to avoid the need for a fixed server by providing functionalities to migrate the distribution engine across the devices involved, when necessary. Connichiwa addresses this issue through the use of local Web servers installed on demand on one of the joined devices. A more peer-to-peer approach was explored in [53], but it was limited to applications obtained through a very specific toolkit (Tcl/Tk), which limits its general applicability.

6.11 Multiple roles

Especially in collaborative environments, characterised by many-to-many device access, it can be important to distribute user interface elements taking into account not only device types, but also user roles. A first contribution addressing this issue is the MU-DUI framework, which provides the possibility of distribution taking into account both device types and users roles, though it must be manually triggered by users according to the possibilities specified by developers.

In ADAM, the authors formalise the problem in terms of optimisation of the assignment of user interface elements to devices taking into account aspects specific to users' roles such as device access, user interface elements permission and privacy, as well as element importance and device characteristics. The optimisation formulation and implementation use existing integer linear solvers. They can automatically search for solutions that maximise the objective function and satisfy the defined constraints while assigning integer solutions to decision variables and give formal bounds on the solution quality with respect to the objective function. Though promising, this approach needs to be empirically validated.

Table 1 summarises how a set of twenty representative approaches addresses the above-mentioned dimensions. Such twenty representative relevant tools or frameworks have been selected in such a way to cover over twenty years of contributions in this area (paying more attention to the most recent proposals); thus, we start from pick-and-drop (1997) until this year (2018, with ADAM). Our aim is to consider contributions published in various venues as well. The table can be used to support different types of analysis. One possible analysis is to compare different proposals in terms of their differences and similarities. For example, pick-and-drop and duet appear to be similar in this design space, and based on the analysis of their values in the dimensions, we can say that they share some limitations: they are not flexible and general tools because they support static distributions of user interfaces with the possibility to migrate only data across devices. They have been useful to point out some interaction possibilities (pick-and-drop was historically one of the first attempts, Duet has a specific focus only on smartwatch/smartphones combinations), but they lack various possibilities that would be useful for developers

of cross-device user interfaces. More generally, we can note that even if in recent years there has been an increasing number of proposal in the field of multi-device UIs, further investigations are necessary in order to better address the insufficiently understood aspects of the logical design space discussed. For example, solutions that exploit peer-to-peer software architectures involving dynamic groups of devices that are accessed by users on the move are still lacking. Another aspect that has been addressed in a limited way, in part due to its complexity, is support for maintaining the state of the running functionalities (e.g. JavaScript functions) associated with the user interface when it migrates across devices. Another topic that deserves further research work concerns general support for a broader group of interaction modality combinations in cross-device user interfaces. This seems an area that should be better explored also considering recent technological evolutions that have made interaction modalities, such as voice and gesture, more widely adopted, and the large number of the various possible combinations of devices and modalities that needs to be supported by general solutions.

Looking carefully at the various possible combinations of values in the various table elements, it is also possible to identify specific aspects not yet sufficiently addressed, such as cross-device frameworks able to support dynamic distribution dependent on user roles or multimodality in user interface distribution. For example, the possibility to distribute user interface elements taking into account different user roles (in addition to the device types) has been addressed in limited ways, as can be seen in the last column of the table. Thus, this can be an area for further investigation, for example with mixed initiative solutions combining automatic distribution recommendations with final user selections.

7 Conclusions and research agenda

In recent years, various systems and tools for providing some level of support for user interfaces in multi-device contexts have been put forward. We have first introduced some relevant concepts, including some multi-device task patterns, and approaches in the area. We have also proposed a classification of contributions in this area in four categories depending on the support provided: responsive, distributed, migratory, and cross-device user interfaces. Then, in order to better compare the features supported by the various environments, we have introduced eleven logical dimensions and discussed how they have been addressed by a representative set of recent proposals. We have also discussed possible areas that deserve future research.

Despite the interest raised, many cross-device interaction techniques and systems require advanced sensing and infrastructure, which are impractical in real-world scenarios outside of the laboratory. For example, the adoption of HuddleLamp [68] poses some limitations on its applicability because it requires that all the devices be located on top of a table that has been equipped with the HuddleLamp technology. This limitation does not promote natural and spontaneous interaction. There are still various issues that prevent the full potential of cross-surface interactions in the wild, for example, related to how users struggle with how devices can communicate, what content can be exchanged, how to opt-out from connecting devices together, and how to configure devices to cooperate in one seamless work-space. We need a better understanding of how to move cross-device techniques and systems into everyday use, in terms of technologies, real-world use cases, and making sense of the available interactions and their impact on human activities.

More generally, a research agenda for this area should consider various aspects. In choosing the dimensions, we considered those that have received some level of attention in the community of designers and developers of multi-device frameworks, as indicated in the relevant publications analysed. Unfortunately, so far, security has received limited attention in this community, and thus, we did not include it as a dimension in the design space. However, we consider such an aspect as deserving of further consideration in the future. Indeed, in order to keep synchronised the user interface parts that are distributed across different interactive devices, they must exchange information. This communication can raise some security and privacy issues. For example, the credit card number in a form should not be visible to people other than the credit card owner, even if the rest of the form is. Moreover, the possibility of distributing user interface elements has to be controlled only by the devices authorised to do so. Indeed, the theft of private information from distributed UIs can occur in various cases involving sensitive information interactively entered (such as credit card, password, etc.), or information included in the accessed page (for example, a bank profile), or data stored in interactive forms, applications sessions, or cookies. A solution to migrate already existing Web applications capable of addressing some security issues such as entering personal private information was proposed [25]. That proposal was based on a migration server, while solutions for distributed architectures supporting cross-device access and interaction would be more general. A proposal for securely distributing user interface parts has been put forward in [1]. It allows developers to specify user interface elements that contain personal private information and cannot be migrated to an unsecure device without the explicit user consent. This can be useful, but probably a solution capable of supporting secure transfer of private information between different devices would be useful as well. A research agenda in this field should consider other topics as well. One such topic is integrated support for several post-WIMP interaction

techniques that can be involved in cross-device user interfaces, such as tangible interaction through physical objects. We should thus consider environments in which user interfaces are distributed at the same time across interactive devices and physical objects. This is something that can also be relevant for various emerging Internet of things applications in domains such as homes, museums, and smart retail. Frameworks and authoring environments able to provide developers with general tools to support easy development of applications fully combining proxemic interaction and cross-device user interfaces would be useful as well, for example by better supporting social interactions taking into account the actual context of use. Some work in this direction has been carried out. For example, Marquardt et al. [50, 51] have developed some specific interaction techniques for small groups arranged according to specific spatial relationships. However, further research is necessary to find general flexible solutions that can be adopted by large communities.

In addition, it would be useful to consider that the increasing availability of devices (see for example the advent of wearables) facilitates the introduction of scenarios in which user interfaces can migrate not only between one device and another, or one device and a group of others, but even between groups of devices.

Another challenge pointed out after interviews with 29 designers or developers of multi-device applications [21] is the lack of tools and methods for testing and debugging cross-device user interfaces. For example, they indicate that some research tools such as Weave and XDStudio provide features to preview multi-device user interfaces, but they lack the ability to simulate usage scenarios with authentic data and contexts. A prototype for a tool supporting emulation of multiple devices and automatic connection management has been put forward in [38], while XD-Testing [39] is a library that provides explicit and implicit device selectors and allows tests to be parameterised by device scenarios that can be randomly generated, but definitive solutions in this area have still not emerged.

Last, but not least, end-user development environments [45] for context-dependent cross-device user interfaces could have a high impact, because we use our applications in ever changing contexts of use with dynamic sets of devices, and in the final analysis, it is the end user who best knows how to adapt to the context and exploit the device richness in order to improve the personal experience. An example in this direction is Improv [11], which applies programming by demonstration in order to obtain cross-device interactions.

References

1. Arthur, R., Olsen, D.R.: Privacy-aware shared UI toolkit for nomadic environments. Softw. Pract. Exp. **42**, 601–628 (2011)
2. Barralon, N., Coutaz, J.: Coupling interaction resources in ambient spaces: there is more than meets the eye! In: Gulliksen, J., Harning, M.B., Palanque, P., van der Veer, G.C., Wesson, J. (eds.) Engineering Interactive Systems. Engineering Interactive Systems, vol. 4940. Springer, Berlin (2008)
3. Barsotti, M., Paternò, F., Pulina, F.: A web framework for cross-device gestures between personal devices and public displays. In: The 16th International Conference on Mobile and Ubiquitous Multimedia (MUM2017), Suttgart, November 2017, pp. 69–78, ACM Press
4. Bellotti, V., Edwards, W.K.: Intelligibility and accountability: human considerations in context-aware systems. Hum. Comput. Interact. **16**(2–4), 193–212 (2001)
5. Bellucci, F., Ghiani, G., Paternò, F., Santoro, C.: Engineering JavaScript state persistence of web applications migrating across multiple devices. In: ACM EICS, pp. 105–110 (2011)
6. Brusilovsky, P.: Adaptive hypermedia. User Model. User-Adapt. Interact. **11**(1–2), 87–110 (2001)
7. Buyukkokten, O., Kaljuvee, O., Garcia-Molina, H., Paepcke, A., Winograd, T.: Efficient web browsing on handheld devices using page and form summarization. ACM Trans. Inf. Syst. Secur. **20**(1), 82–115 (2002)
8. Cantera, J.M.: Model-Based UI XG Final Report https://www.w3.org/2005/Incubator/model-based-ui/XGR-mbui-20100504/ (2010)
9. Chang, T. H., Li, Y.: Deep shot: a framework for migrating tasks across devices using mobile phone cameras. In: Proceedings ACM CHI'11, pp. 2163–2172
10. Chen, X. A., Grossman, T., Wigdor, D. J., Fitzmaurice, G.W.: Duet: exploring joint interactions on a smart phone and a smart watch. In: CHI 2014, pp. 159–168
11. Chen, X., Li, Y.: Improv: an input framework for improvising cross-device interaction by demonstration. ACM Trans. Comput. Hum. Interact. **24**(2), 15 (2017)
12. Cheverst, K., Davies, N., Mitchell, K., Efstratiou, C.: Using context as a crystal ball: rewards and pitfalls. Pers. Ubiquitous Comput. **5**(1), 8–11 (2001)
13. Chi, P. Y. P., Li, Y.: Weave: scripting cross-device wearable interaction. In: ACM CHI 2015, Seoul, Korea, April 2015, pp. 3923–3932
14. Corporate Act-Net Consortium.:The active database management system manifesto: a rulebase of ADBMS features. In: ACM SIGMOD Record, vol. 25 (1996)
15. Coutaz, J., Lachenal, C., Dupuy-Chessa, S.: Ontology for multi-surface interaction. In: Proceedings of IFIP Conference on Human-Computer Interaction Interact'2003, IOS Press, Amsterdam, pp. 447–454 (2003)
16. Coutaz, J., Nigay, L., Salber, D., Blandford, A., May, J., Young, R.: Four easy pieces for assessing the usability of multimodal interaction: the CARE properties. In: Proceedings INTERACT, pp. 115–120 (1995)
17. Dearman, D., Pierce, J.: It's on my other Computer!: computing with multiple devices. In: Proceedings of CHI'08, ACM Press, pp. 767–776 (2008)
18. Demeure, A., Sottet, J. S., Calvary, G., Coutaz, J., Ganneau, V., Vanderdonckt, J.: The 4C reference model for distributed user interfaces. In: Fourth International Conference on Autonomic and Autonomous Systems, ICAS 2008, pp. 61–69, (2008)
19. Dey, A.: Understanding and using context. Pers. Ubiquit. Comput. **5**(1), 4–7 (2001)
20. Di Geronimo, L., Bertarini, M., Badertscher, J., Husmann, M., Norrie, M. C.: Exploiting mid-air gestures to share data among devices. In Proceedings of the 19th International Conference on Human-Computer Interaction with Mobile Devices and Services (MobileHCI '17), vol. 35 (2017)

21. Dong, T., Churchill, E., Nichols, J.: Understanding the challenges of designing and developing multi-device experiences. In: Proceedings DIS 2016, pp. 62–72. ACM Press
22. Fischer, C., Gellersen, H., Gostner, R., Guinard, D., Kortuem, G., Kray, C., Rukzio, E., Strengin, S.: Supporting device discovery and spontaneous interaction with spatial references. Mobile Spat. Interact. Pers. Ubiquitous Comput. (2008). https://doi.org/10.1007/s00779-008-0206-3
23. Frosini, L., Paternò, F.: User interface distribution in multi-device and multi-user environments with dynamically migrating engines. ACM EICS **2014**, 55–64 (2014)
24. Gajos, K.Z., Weld, D.S., Wobbrock, J.O.: Automatically generating personalized user interfaces with Supple. Artif. Intell. **174**(12), 910–950 (2010)
25. Ghiani, G., Isoni, L., Paternò, F.: Security in migratory interactive web applications. In: Proceedings of the 11th International Conference on Mobile and Ubiquitous Multimedia MUM 2012, vol. 15. ACM Press (2012)
26. Ghiani, G., Manca, M., Paternò, F., Porta, C.: Beyond responsive design: context-dependent multimodal augmentation of web applic. In: MobiWIS 2014, LNCS, vol. 8640, pp. 71–85. Springer (2014)
27. Ghiani, G., Paternò, F.: Supporting mobile users in selecting target devices. J. Univ. Comput. Sci. **16**(15), 2019–2037 (2010)
28. Ghiani, G., Paternò, F., Santoro, C.: Push and pull of web user interfaces in multi-device environments. In: AVI 2012, pp. 10–17
29. Ghiani, G., Paternò, F., Santoro, C.: Interactive customization of ubiquitous Web applications. J. Vis. Lang. Comput. **24**(1), 37–52 (2013)
30. Ghiani, G., Polet, J., Antila, V.: Towards intelligent migration of user interfaces. In: MobiWIS 2013, pp. 203–217 (2013)
31. Ghiani, G., Manca, M., Paternò, F., Santoro, C.: Personalization of context-dependent applications through trigger-action rules. ACM Trans. Comput. Hum. Interact. **24**(2), 14 (2017)
32. Gjerlufsen, T., Klokmose, C. N., Eagan, J., Pillias, C., Beaudouin-Lafon, M.: Shared substance: developing flexible multi-surface applications. CHI 2011, pp. 3383–3392 (2011)
33. Google Research Report.: The New Multi-screen World: Understanding Cross-Platform Consumer Behavior. http://services.google.com/fh/files/misc/multiscreenworld_final.pdf (2012). Accessed 3 April 2019
34. Greenberg, S., Boring, S., Vermeulen, J., Dostal, J.: Dark patterns in proxemic interactions: a critical perspective. In Proceedings of DIS'14, pp. 523–532, ACM
35. Hamilton, P., Wigdor, D.J.: Conductor: enabling and understanding cross-device interaction. In Proceedings of CHI'14, pp. 2773–2782. ACM Press (2014)
36. Houben, S., Marquardt, N.: WatchConnect: a toolkit for prototyping smartwatch-centric cross-device applications. In: CHI 2015, pp. 1247–1256 (2015)
37. Houben, S., Tell, P., Bardram, J.E. Bardram: ActivitySpace: managing device ecologies in an activity-centric configuration space. In: ITS 2014, pp. 119–128 (2014)
38. Husmann, M., Nebeling, M., Pongelli, S., Norrie, M.C.: MultiMasher: providing architectural support and visual tools for multi-device mashups. WISE **2**(2014), 199–214 (2013)
39. Husmann, M., Spiegel, M., Murolo, A., Norrie, M. C.: UI testing cross-device applications. In: Proceedings of the 2016 ACM on Interactive Surfaces and Spaces, ISS '16, pp. 179–188. New York. ACM (2016)
40. Johanson, B., Ponnekanti, S., Sengupta, C., Fox, A.: Multibrowsing: moving web content across multiple displays. In: Proceedings of Ubicomp 2001, Springe, LNCS, vol. 2201, pp. 346–353 (2001)
41. Johanson, B., Fox, A., Winograd, T.: The Interactive Workspaces Project: Experiences with Ubiquitous Computing Rooms. IEEE Pervasive Comput. **1**(2), 67–74 (2002)
42. Jokela, T., Ojala, J., Grassel, G., Piippo, P., Olsson, T.: A comparison of methods to move visual objects between personal mobile devices in different contexts of use. In: Proceedings of MobileHCI'15, pp. 172–181
43. Jokela, T., Ojala, J., Olsson, T.: A diary study on combining multiple information devices in everyday activities and tasks. In: CHI 2015, pp. 3903–3912 (2015)
44. Kulkarni, C., Klemmer, S.: Automatically adapting web pages to heterogeneous devices. In: CHI'11 Extended Abstracts on Human Factors in Computing Systems (CHI EA'11). ACM, New York, pp. 1573–1578 (2011)
45. Lieberman, H., Paternò, F., Klann, M., Wulf, V.: End-user development: an emerging paradigm. In: Lieberman, H., Paternò, F., Wulf, V. (eds.) End-user Development. Human-Computer Interaction Series, pp. 1–8. Springer, Berlin (2006)
46. Lo, J., Wohlstadter, E., Mesbah, A.: Imagen: runtime migration of browser sessions for javascript web applications. In: Proceedings of the International World Wide Web Conference (WWW), pp. 815–825 (2013)
47. Manca, M. Paternò, F.: Customizable dynamic user interface distribution. In: Proceedings ACM EICS'16, Bruxells, pp. 27–37. ACM Press
48. Marcotte, E.: Responsive Web Design, A Book Apart. http://www.abookapart.com/products/responsive-web-design (2011). Accessed 3 April 2019
49. Marquardt, N., Diaz-Marino, R., Boring, S., Greenberg, S.: The proximity toolkit: prototyping proxemic interactions in ubiquitous computing ecologies. In: Proceedings of UIST 2011, pp. 315–326. ACM (2011)
50. Marquardt, N., Hinckley, K., Greenberg, S.: Cross-device interaction via micro-mobility and f-formations. In: Proceedings of the 25th Annual ACM Symposium on User Interface Software and Technology, pp. 13–22. ACM Press
51. Marquardt, N., Ballendat, T., Boring, S., Greenberg, S., Hinckley, K.: Gradual engagement: facilitating information exchange between digital devices as a function of proximity. In: Proceedings of the 2012 ACM International Conference on Interactive tabletops and surfaces, pp. 31–40. ACM Press
52. Melchior, J., Grolaux, D., Vanderdonckt, J., Van Roy, P.: A toolkit for peer-to-peer distributed user interfaces: concepts, implementation, and applications. In: Proceedings of EICS 2009, pp. 69–78. ACM, (2009)
53. Melchior, J., Vanderdonckt, J., Van Roy, P.: A model-based approach for distributed user interfaces. In: Proceedings of the 3rd ACM SIGCHI Symposium on Engineering Interactive Computing Systems (EICS '11). ACM, pp. 11–20 (2009)
54. Meskens, J., Vermeulen, J., Luyten, K., Coninx, K.: Gummy for multi-platform user interface designs: shape me, multiply me, fix me, use me. In: Proceedings of the Working Conference on Advanced Visual Interfaces (AVI'08), pp. 233–240. ACM
55. Myers, B., Hudson, S.E., Pausch, R.: Past, present, and future of user interface software tools. ACM Trans. Comput. Hum. Interact. **7**(1), 3–28 (2000)
56. Nacenta, M.A., Aliakseyeu, D., Subramanian, S., Gutwin, C.: A comparison of techniques for multi-display reaching. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '05). ACM, New York, pp. 371–380 (2005). http://dx.doi.org/10.1145/1054972.1055024
57. Nebeling, M.: XDBrowser 2.0: semi-automatic generation of cross-device interfaces. In: Proceedings CHI 2017, pp. 4574–4584. ACM Press
58. Nebeling, M., Husmann, M., Zimmerli, C., Valente, G., Norrie, M. C.: XDSession: integrated development and testing of cross-device applications. In: EICS 2015, pp. 22–27 (2015)

59. Nebeling, M., Mintsi, T., Husmann, M., Norrie, M.C.: Interactive development of cross-device user interfaces, pp. 2793–2802. ACM CHI, Norrie (2014)
60. Nebeling, M., Speicher, M., Norrie, M. C.: CrowdAdapt: enabling crowdsourced web page adaptation for individual viewing conditions and preferences. In: EICS 2013, pp. 23–32 (2013)
61. Park, S., Gebhardt, C., Rädle, R., Feit, A.M., Vrzakova, H., Dayama, N.R., Yeo, H.S., Klokmose, C.N., Quigley, A., Oulasvirta, A., Hilliges, O.: AdaM: adapting multi-user interfaces for collaborative environments in real-time. In: Proceedings ACM CHI 2018, Monterey. ACM Press (2018)
62. Paternò,F., Santoro, C.: A logical framework for multi-device user interfaces. In: EICS 2012, pp. 45–50, ACM Press,Copenaghen (2012)
63. Paternò Santoro, C., Spano, L.D.: MARIA: a universal language for service-oriented applications in ubiquitous environment. ACM Trans. Comput. Hum. Interact. **16**(4), 191–1930 (2009)
64. Paternò, F., Santoro, C., Spano, L.D.: Engineering the authoring of usable service front ends. J. Syst. Softw. **84**(10), 1806–1822 (2011)
65. S. Ponnekanti, B. Lee, A.Fox, and P.Hanrahan, Icrafter: A service framework for ubiquitous computing environments. Proc. ACM Ubiquitous Computing (UbiComp'01) (Jan 2001), pp.56–75
66. Ponnekanti, S. R., Johanson, B., Kiciman, E., Fox, A.: Portability, extensibility and robustness in iROS. In Proceedings of the 1st IEEE International Conference on Pervasive Computing and Communications (PERCOM '03)
67. Rädle, R., Jetter, H. C., Marquardt, N., Reiterer, H., Rogers, Y.: HuddleLamp: spatially-aware mobile displays for ad-hoc around-the-table collaboration. In Proceedings ITS'14, pp. 45–54. ACM (2014)
68. Rädle, R., Jetter, H. C., Schreiner, M., Lu, Z., Reiterer, H., Rogers, Y.: Spatially-aware or spatially-agnostic?: Elicitation and evaluation of user-defined cross-device interactions. In: CHI, pp. 3913–3922 (2015)
69. Rekimoto, J.: Pick-and-drop: a direct manipulation technique for multiple computer environments. In: ACM Symposium on User Interface Software and Technology, pp. 31–39 (1997)
70. Rekimoto J., Saitoh, M.: Augmented surfaces: a spatially continuous work space for hybrid computing environments. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '99). ACM, New York, pp. 378–385 (1999). Doi: http://dx.doi.org/10.1145/302979.303113
71. Sanctorum, A. Signer, B.: Towards end-user development of distributed user interfaces. In: UAIS, pp. 1–15. Springer (2017)
72. Santosa, S., Wigdor, D.: A field study of multi-device workflows in distributed workspaces. 2013. A field study of multi-device workflows in distributed workspaces. In: Proceedings of the 2013 ACM International Joint Conference on Pervasive and Ubiquitous Computing (UbiComp'13). ACM, New York, pp. 63–72
73. Schreiner, M., Rädle, R., Jetter, H. C., Reiterer, H.: Connichiwa: a framework for cross-device web applications. In: CHI Extended Abstracts 2015, pp. 2163–2168
74. Serrano, M., Ens, B., Yang, X. D., Irani, P: Developing a head-worn display interface to unify the interaction experience in distributed display environments. In: Proceedings of MobileHCI'15, pp. 16–171. ACM Press
75. Sohn, T., Li, F.C.Y., Battestini, A., Setlur, V., Mori, K. Horii, H.: Myngle: unifying and filtering web content for unplanned access between multiple personal devices. In: Proceedings UbiComp 2011, pp. 257–266. ACM
76. Tandler, P., Prante, T., Müller-Tomfelde, C., Streitz, N. and Steinmetz, R.: ConnecTables: dynamic coupling of displays for the flexible creation of shared workspaces. In: Proceedings of 14th ACM Symposium on UI Software and Technical UIST'01,, pp. 11–20. ACM Press, New York (2001)
77. Terrenghi, L., Quigley, A., Dix, A.: A taxonomy for and analysis of multi-person-display ecosystems. Pers Ubiquitous Comput **13**(8), 583–598 (2009)
78. Vandervelpen, C., Conix, K.: Towards model-based design support for distributed user interfaces. In: Proceedings of NordiCHI 2004, pp. 61–70. ACM (2004)
79. Wäljas, M., Segerståhl, K., Väänänen-Vainio-Mattila, K., Oinas-Kukkonen, H.: Cross-platform service user experience: a field study and an initial framework, In: Proceedings of MobileHCI 2010 Lisboa, Portugal, September 2010, pp. 219–228. ACM Press (2010)
80. Yang, J. Wigdor, D.: Panelrama: enabling easy specification of cross-device web applications. In: ACM CHI 2014, pp. 2783–2792, Toronto (2014)