Delegation Theory in the Design of Cross-Platform User Interfaces

Dagmawi L. Gobena[1(✉)], Gonçalo N.P. Amador[2], Abel J.P. Gomes[3], and Dejene Ejigu[1]

[1] Addis Ababa University, Addis Ababa, Ethiopia
dagmawi.Lemma@gmail.com, ejigud@yahoo.com
[2] University of Beira Interior, Covilhã, Portugal
gamador@it.ubi.pt
[3] Instituto de Telecomunicações, Covilhã, Portugal
agomes@di.ubi.pt

Abstract. The amalgamation of various technologies to support the needs of new computing models has become prevalent in computing environments like ubiquitous computing. Amalgamation means here heterogeneity caused by not only the coexistence of various devices in the same computing environment, but also the diversity between software, users as well as interaction modalities. The platform heterogeneity together with additional needs of interaction modalities and the proliferation of new technologies pose unique challenges for user interface (UI) designers and developers. We consider the problem of heterogeneity as a demand of collaboration between platforms (device and system) that are owned or controlled by a human user. Hence, we drive the concept of delegation to be implemented in a peer-to-peer model, where one peer (known as *delegator*) delegates another peer (known as *delegatee*) to run a UI (or a single interaction-modality) on its behalf. Thus, the delegatee uses its own capabilities to present the required UI or interaction-modality.

Keywords: Cross-platform UI · UI migration · Distributed UI

1 Introduction

Delegation is the act of appointing others in order to discharge a certain responsibility on behalf of the appointer. The concept of delegation may appear in various forms and within diversified context. Nevertheless, in any form of delegation, there are two parties – the *delegator* and the *delegatee*. The former empower or appoint the later to discharge a task on its behalf.

Accordingly, we argue that, an interaction might be held across a platform if a device is appointed to run the user interface (UI) on behalf of another device while the service of the task runs at the delegator end. This approach is similar to some existing services such as remote desktop. But in remote desktop, identical interaction modality is considered between peers. We include the possibilities of assimilating diverse interaction modalities using peer-to-peer model. Each peer is autonomous to apply the interaction modalities as per their capabilities. For example, a visual modality might be converted

M. Kurosu (Ed.): Human-Computer Interaction, Part II, HCII 2015, LNCS 9170, pp. 519–530, 2015.
DOI: 10.1007/978-3-319-20916-6_48

into voice interaction at the peer delegated to run a certain UI. Thus, in this paper, we extend the concept of delegation to be applied in developing cross-platform UI within ubiquitous computing, which is characterized by heterogeneity of diverse elements.

The heterogeneity is caused by the coexistence of various devices in the same computing environment as well as due to the diversity between software, users, interaction modalities and environments.

It happens that two of the prominent approaches (i.e., UI distribution and UI migration), usually followed in the development of cross-platform UI, focus on a particular aspect of the heterogeneity – mostly the device [2]. But, while having heterogeneous environment, if the UI is generated based on a specific context (e.g., the device, user, task, interaction modalities, etc.), it is likely that the usability of the system would be reduced entailing several usability issues [1]. Furthermore, since both UI distribution and UI migrations are often based on client-server model, the entire cross-platform UI environment is subjected to single point of failure.

Rather, we consider the problem of heterogeneity as a demand of collaboration between platforms (device and system) that are part of the computing environment (e.g., wearable platform like a smart watch) or controlled by a human user.

We are motivated to develop the theory of *UI delegation* to sustain an alternative approach in the development of cross-platform UI within a peer-to-peer model where one peer (known as *delegator*) delegates another peer (known as *delegatee*) to run a UI/interaction-modality on its behalf. This means that the *delegatee* is autonomous and uses its own capabilities to present the required UI/interaction-modality. In other words, the UI (resp., interaction modality) generated (resp., used) by the delegatee does not need to be identical to delegator's one, but has to perform the function to achieve the same goal. Thus, we consider the concept of *UI delegation* as an opportunity to take advantage of heterogeneity, so that interaction can be extended and usability can be improved using the capabilities of the *delegatee*. For example, a list box widget can be used "on behalf of" radio button for implementing "choice" concept in the interaction. Similarly, instead of visually reading a text from the screen it can be converted to audio and played if the capability exists. Thus, audio listening can be used "on behalf of" of visual reading.

Summing up, the theory of *UI delegation* is presented in this paper to take advantage of heterogeneity inherent to different devices, systems, users, interaction modalities co-existing in a ubiquitous environment. In Sect. 2, we briefly review the literature related to concepts of cross-platform UI and types of user interfaces. The main constructs of theory of *UI delegation* are then discussed in Sect. 3. In Sect. 4, we discuss the *common interface language* (CIL) as a protocol for realizing *UI delegation* within the context of heterogeneous environment. Finally in Sect. 5 we made our conclusion.

2 Literature Review

Ubiquitous computing comprises various platforms to achieve its goal. These platforms are heterogeneous in type, technology, as well as function. For example, browsing the Internet is a common function that can be supported by various computing devices in

the same environment, but such devices make usage of different browsers, display sizes, screen resolutions, operating systems, etc. – even by different users and in different contexts. Because of this, one of the characteristics of ubiquitous computing is *heterogeneity*. [1] In this setting, users interact with devices implicitly or explicitly using different interaction modalities [2].

The heterogeneity of the platforms can be felt at various levels and from different perspectives: users, UI designers, as well as the usability of the system. With respect to users, they are required to learn new UI and interaction modalities. Regarding the usability of the system, the learning curve of the user could take more time until the user becomes adequately proficient and efficient in dealing with the system. From designers' perspective, they may be required to develop unique UIs for each platform [3].

Vanderdonckt [4] classifies the UI design for heterogeneous platforms as per the situation that causes the diversity. Therefore, the UI design may focus on the presence of multiple users or, alternatively, on the usage of multiple monitors, devices, platforms, and displays [5].

Nevertheless, designing various versions of same UI for distinct platforms operating within the same environment entails several problems. Foremost, there would be duplication of effort in the UI development process and also.

- changing/adding/removing features consistently across each platform could be cumbersome [6], and this leads us to the problem of *maintainability* of each type of platform;
- introducing new platform may require rework, since it is important to have new design for supporting the newly added platform; and this would challenge the *scalability* of the environment.

Another approach in cross-platform UI development is adapting a UI for the context of a platform by automatically *generating* the UI or, alternatively, *migrating* UI [7–9]. In this regards, UI distribution and UI migration have become two of the prominent approaches. These approaches are followed as general UI development approaches [10–14]. In [14], distributed and migratory UIs are discussed as two independent concepts.

In UI distribution, UI elements are distributed across platforms, so that, in some cases, this may create duplication of UI elements [14]. For example, in [15], a multi-client (multi-platform) UI is presented using the model-view-controller (MVC) architecture that stores different versions of a webpage (UI) on the server for each predefined platform; and a controller is responsible for selecting one of the UI versions that most fits a particular platform from which the users operate. Also, UI distribution depends on predefined UI elements created at design time and available (or distributed across platforms) with a certain platform/user context in mind.

The action of transferring a UI from one device (source) to another device (sink) is called UI migration; the UI itself is known as migratory UI, which is "said to be *migratable* if it has the migration ability" [16]. Migratory ability is the ability of a UI element to be rendered at a remote peer. Hence, UI designers can decide on the part of the UI that shall be migrated across platform as the need arise. Thus a UI can be migrated partially or completely [10]. However, the interaction modality between the peers remains the same. Hence mono-modality is often followed [10].

Migrating UI can help a user to continue computing on the go. Thus, unlike distributed UIs, during migration not only the structure of the UI element but also their content is maintained and migrated [10, 14]. For example, if a user selects a checkbox on an interface before migration, then that checkbox shall appear as selected checkbox across a platform where the UI is migrated and rendered.

Migratory UIs can be seen as a particular type of distributed UI. Berti et al. [10] indicated that a UI migration process is mainly started on-demand. That is, if the user desires to continue the interaction on a different platform, but using exactly the same UI.

What distributed and migratory UI have in common is that they both run over the client-server computing architecture. Consequently, in addition to the unsolved problems of *scalability* and *maintainability*, these UIs suffer from lack of reliability because the UI server is a single point of failure.

In our opinion, the current cross-platform UI development methods need a different dimension, where a UI can be automatically generated from runtime UI related information about the device, the system and human user. It is here that delegation theory and delegated UIs come into play.

The notion of delegation is applied in some areas of computing. For example in [17], devices with less computing power benefits the computational power of delegated devices with higher computational power for assigning and revoking privileges to users in pervasive environment. Haddadi also developed a theory, in agent-based system, by taking "an internal perspective to model how individual agents may reason about their actions" [18]. This is further developed in [19], where it is stated that "in delegation an agent A needs or likes an action of another agent B and includes it in its own plan, thus, A is trying to achieve some of its goals through B's action". According to Castelfranchi et al., A is said to be the "client", while B is the "contractor" [19].

Since heterogeneous environment introduce several diversified platform capabilities at each distinct peer, this opportunity shall be consider in improving usability of the system by letting the user to interact through collaborated capabilities. Hence, we foresee a different dimension of cross-platform UIs that can be realized by delegating a peer to run a UI on behalf of another peer. For example, it is more convenient to compose a message on the standard keyboard than on virtual keyboard of small handheld device. Thus, thought the actual messaging service (e.g., SMS) can be delivered via a mobile phone, for instance, the user could be more satisfied if allowed to interact with its standard keyboard for the purpose of composing the message. Hence, the usability can be improved if the mobile phone delegates the desktop only for the HCI aspect of the messaging system.

According to the works and techniques we found in the literature, at the time of automatically generating a specific UI across platform within a certain computing environment, they focus only on one of the views (user, device or system). Though it is common practice to consider the user and platform capabilities in UI development at design time [20], this is not the case at runtime. For example, responsive web development approaches as well as solutions presented by Zhang et al., in their pattern based approach [7], the focus is on screen size adaptation. On the other hand, Jeffery et al. [6] focused on the functionality of the appliances while Sauter et al. focused [15] only consider the device type, just to mention a few works.

Therefore, we hypothesize that if each peer within the environment describe its UI capabilities and store it locally, advertising or making them available to other peers using the same protocol (i.e. CIL) whenever needed, and if there is at least another peer (*delegatee*) that matches its capabilities literally or by transmutability, then it is possible to run the UI or interaction modality on behalf of the *delegator.*

In this paper, delegating the build-ups of a UI on some device in the computing environment means that we are not using the client-server computing model anymore, but the peer-to-peer computing model. So,

1. since each peer is responsible to maintain and locally store its own capabilities, it is always possible to authorize new peers signing up and in the environment; i.e., the environment can be scaled up easily,
2. since the desired UI or interaction modality is generated during interaction, there is no need to create the UI element at design time (e.g., a menu to access a certain functionality), thus the environment would be easy to maintain.

Moreover, we assume a holistic approach of the UI by considering triplet views (the user, device and system) in cross-platform UI, but not in partiality of any of the views. This is especially useful when the peers communicate their capabilities during the delegation process. Thus, delegated UIs sustain themselves on *autonomy* of the delegatee, as well as on *transmutability* of capabilities between the delegator and the delegatee.

3 Theory of UI Delegation

There are two decision makers in delegation: the principal (*delegator*) and the agent (*delegatee*). The *delegator* decides whether to initiate the delegation or not while the *delegatee* may be willing to participate in the delegation process or not [21]. Thus, both the *delegator* and *delegatee* are autonomous. Hence, *UI delegation* shall take place within a peer-to-peer model.

In *UI delegation,* the *delegatee* may initiate the delegation process on-demand or automatically. The delegation process can be initiated on-demand if the user wants to continue the interaction with the system but using another peer possibly owning more suitable capabilities. For example, a user who wants to compose an email that would be sent from its smartphone may wish to use a standard keyboard attached to his desktop computer. Hence the user shall initiate the delegation process manually so that the smartphone may delegate the desktop. Likewise, automatic delegation can be initiated by the *delegator* if a peer believes it is deemed important to do so. For example, in a situation where a user interacts within a context-aware system using the speech/audio modality at a desktop in his/her office, once the user changes the location and enters in to a noisy zone, a peer may decide to switch to the visual modality while delegating the UI on handheld device so that the user may continue the interaction using visual modality, providing that the *delegatee* peer is dressed up with the desired capabilities.

On the other hand, the *delegatee* peer might be willing or not to participate in the delegation process. Furthermore, considering a heterogeneous environment where peers are dressed up with various capabilities, and are able to support diversified

functionalities, not all the peers shall participate in *an instance of a delegation process*. Accordingly, the peers can be classified as *delegatee*, *candidate* or neither. Thus, in the delegation process:

- A peer is said to be *delegatee* if it responds to a *delegation request* and necessarily selected by a *delegator* peer
- A peer is said to be *candidate peer* when it responds to a *delegation request*

Therefore, a reply to a *delegation request* indicates that the responding peer is willing to take part in the delegation process.

In order to participate in *UI delegation,* each peer shall describe its supported capabilities and store them locally. This can be done at design time and modified as required to incorporate new capabilities or removing undesired ones. However, selection of capabilities and rendering the UI accordingly can be done at runtime. The local description is then used when a peer creates and transmits a *delegation request*, and when a peer checks the degree of matching of its capabilities with other peer. The former is responsibility of the *delegator*, while the latter is responsibility of the *delegatee.*

3.1 Delegation Request

A peer willing to delegate another peer shall create a *delegation request* to peers within the same environment in order to be aware of existing capabilities available across such environment. Thus, a *delegation request* conveys a partial description of a UI or interaction modalities in terms of the desired capabilities required to build a usable interaction. Therefore, the *delegator* peer shall describe the required capabilities and then send them as a *delegation request* for each peer in the environment. At this stage, only basic information about the UI to be delegated is required.

For example, if there is a selection widget in the prospective UI to be delegated, then only its basic information is required, but not the details, such as the content (value) of the widget and the data structure (data type). As explained further ahead, delegation requests and communication between peers can take place using a XML-based protocol.

3.2 Degree of Matching

Having the *delegation request* in the form of XML description, each *candidate peer* shall compute the degree of matching *M* between capabilities, which is used to determine how much a peer's capability resembles to what is requested by the *delegator* peer. In this regards, three possible conditions can be considered:

1. There is identical capability
2. There is similar but not identical capability
3. The requested capability is not available.

In the case that the capability is identically supported by the *candidate peer*, the value of M is incremented by 2; if the homologous capabilities are similar, the value of M is incremented by 1; otherwise, the capability does not exist on the candidate delegatee side, even using transmutation, so the value of M shall be decremented to discourage the competing *candidate peer* so as to minimize the chance of appointment as *delegatee.* Thus, after computing the value of M, each *candidate peer* P_i returns $M_i$to delegator as its value of M. Therefore, for a given list of capabilities in the *delegation request* with size N, the maximum possible value that P_i can attain (i.e. if all the capabilities in the *delegation request* identically matches the corresponding local capabilities maintained by the *candidate peer*) is given by:

$$M = 2N \tag{1}$$

On the other hand, if all the capabilities in the *delegation request* can be supported by the *candidate peer,* using similar capabilities but not identical capability, then the value of M would be:

$$M = N \tag{2}$$

Therefore, suppose the number of candidate peers is n, then it is most appropriate to select a peer with the highest *degree of matching,* which is given by

$$\text{MAX } (M_1, M_2, \ldots, M_n) : 1 < i < n \text{ and } N \leq M_i \leq 2N \tag{3}$$

Thus, the *UI delegation* holds:

- if there exists an active peer that responds to *delegation request*, and
- if Eq. (3) is satisfied.

Nevertheless, Eq. (3) is subjected to design decision and the UI developers might decide letting the delegation to prevail if $N < M$, but if $N \approx M$ variation may occur with respect to Eq. (3).

3.3 Flow of *UI Delegation*

The following flow of steps intends to clarify the idea of *UI delegation* that is presented as within a heterogeneous environment.

1. user starts operation at platform p_i $(i = 1, 2, \ldots, n)$, *wheren* is the number of platforms within the environment;
2. after a while, the user changes its location;

3. platform p_i is aware of the new user location (for example, the absence of UI can be traced/anticipated if the user is not responsive in a certain given time, and location service can be then used to learn about the new location of the user)
4. platform p_i broadcasts *delegation request* message $M_{pi,}$ given by Eq. 4

$$M_{pi} = \{d_i, h_i, s_i\} \tag{4}$$

where d, h and s stand for the desired device, human and system capabilities respectively, that should be considered while applying the UI or interaction modality at the delegatee end;

5. platforms within the environment *advertise* their *degree of matching* with M_{pi};
6. platform p_i delegates device p_j since p_j better supports the desired capabilities;
7. platform p_j uses the description of M_{pi} to generate as per the locally maintained capability

$$M_{pj} = \{d_j, h_j, s_j\} \tag{5}$$

8. user operates at platform p_j using the delegated UI.

Therefore, since not all platforms in the environment are the most adequate for the delegation, the respondent peer that most fits the required capability will become the delegatee.

4 The CIL Protocol

In order to maintain the collaboration between peers, as well as to standardize how capabilities are represented and exchanged between peers, the peers shall use the CIL (Common Interface Language) as a protocol for describing: the *capabilities*, the *presentation*, and the *message to be exchanged between the peers.*

Furthermore, each description shall abide the syntax and semantics as defined in the *CIL-definition.*

We create *CIL-definition* using XML-schema for setting the constraints and structure of the descriptions that would be used between the peers. Hence, description for representing the UI, message, and capabilities shall be validated as per the *CIL-definition.*

The *CIL-definition* is created using three broad aspects deemed to be important in any HCI: the *human*, *device* and *system*. As indicated in the following fragment of the XML-schema, these three views are represented as the top-level elements of a taxonomy for cross-platform interfaces, so that any description shall be created under either of these elements.

```
<complexType name="CIL">
  <all>
    <element name="_HumanBeingView"
  type="xs:_HumanBeingView"
    maxOccurs="1" minOccurs="0">
</element>
<element name="_DeviceBeingView"
  type="xs:_DeviceBeingView"
    maxOccurs="1" minOccurs="0">
</element>
<element name="_SystemBeingView"
  type="xs:_SystemBeingView"
      maxOccurs="1" minOccurs="0">
</element>
  </all>
<attribute name="Version" type="string"/>
<attribute name="id" type="string"/>
</complexType>
```

The `_HumanBeingView` and `_DeviceBeingView` are used to define the required elements and respective constraints useful to describe human and device capabilities. For example, a human capability is the eye sight, so we need to specify whether or not a given individual is trichromat or colorblind.

```
<xs:_HumanBeingView>
  <xs:Sensory>
    <xs:Vision Supported="true" Preference="1">
      <xs:ColorBlind>true</xs:ColorBlind>
    </xs:Vision>
  </xs:Sensory>
  <xs:Effector Supported="true" Preference="2">
    <xs:Touching Supported="true" Preference="0">
        <xs:ArmCount>0</xs:ArmCount>
    </xs:Touching>
  </xs:Effector>
</xs:_HumanBeingView>
```

Thus, child elements are created under each of those top-level elements in the schema, which are useful to describe and represent:

- the human physical interaction capabilities;
- the device interaction modalities, with a focus on the input and output mechanisms possibly supported by the device

However, it is not important to create an exhaustive description of capabilities using all the elements defined within the schema. Rather, each peer shall locally describe and store its own capabilities using the same schema, so that such list of capabilities can be used.

- when determining the *degree of matching M* after receiving a *delegation request.*
- to describe UI that to be run at a certain *delegatee* end

For example, if a *delegator* desires to run a UI that is meant to support a colorblind user, then part of the *delegation request* description shall include this information under the `_HumanBeingView`.

The presentational description (or description of the running UI/interaction modalities) has to be created in accordance to the elements and the respective constraints set under `_SystemBeingView`. Nevertheless, the elements under this element can be used to describe platform capabilities related to services useful for HCI as well. For example, a text-to-speech conversion capability that may exist at one of the peer can be described under the `_SystemBeingView` within the `presentation` element.

The extent how CIL is used might vary at different level of the delegation process. For example, though we include all the possible generic information that can be required and useful to describe UIs or interaction modalities using XML-schema, it may not be important to use the entire structure; for instance for *delegation request*.

```
<complexType name="Widget">
  <sequence>
  <element name="WidgetClass" type="xs:WidgetClass"
      maxOccurs="1" minOccurs="1">
  </element>
    <element name="WidgetStructure"
  type="xs:WidgetStructure"
      maxOccurs="1" minOccurs="1">
  </element>
  <element name="WidgetContent" type="xs:WidgetContent"
      maxOccurs="unbounded" minOccurs="0">
  </element>
  <element name="WidgetState" type="xs:WidgetState"
      maxOccurs="1" minOccurs="0">
  </element>
  <element name="WidgetSize" type="xs:WidgetSize"
      maxOccurs="1" minOccurs="0">
  </element>
</sequence>
<attribute name="WidgetID" type="string"></attribute>
<attribute name="Identifier" type="string"></attribute>
</complexType>
```

For example the above schema can be useful to describe the basic information about a particular widget that severs as a build-up of the UI. In this case, the `Widget` is an element that is defined under the `WidgetSet`, which is used to represent the form of explicit interaction. And the `Widget` element is further described by its class, using the `WidgetClass` element. Other parts of the schema shown in the above code are useful for defining the presentational description, but can be less relevant within the

delegation request. Therefore, each peer within the environment, where *UI delegation* is applied, has to describe its capability using as per the *CIL-definition* and store it locally; and when required a peer can advertise its capability in the form of *delegation request* or for computing the *degree of matching* in response to *delegation request.*

It is important to note that the entire schema we created for CIL can be updated and improved by incorporating new elements to include new UI style or interaction modalities. But peers participating in the *UI delegation* process shall describe their capabilities and store is locally using the same version of the schema, known as *CIL-definition.* At this stage of our study, four classes of widget are identified: `selection`, `text`, `view` and `switch`. However, it has to be noted that this classes may appear in any form across various platforms. For example, choice can be presented in the form of radio, checkbox, list box, etc. Thus what is needed during *delegation request* is the basic information that shows whether the candidate peer supports the `selection` class or not.

5 Conclusion

In the *theory of UI delegation* we have identified four constructs. The concept of CIL is the main construct that is useful to realize the communication and collaboration between peers. Yet, the peers may appear as *delegator, delegatee* or *candidate* peer. Thus peer type is the second construct in the theory. However, the distinction between *delegatee* and *candidate* peer related to two other important constructs: *delegation request* and *degree of matching.* We argue that it is possible to generate at runtime a more usable UI across platforms as well as build scalable and maintainable heterogeneous environment by following the theory of *UI delegation*. Nevertheless, delegation is based on trust, thus more work has to be done regarding trust management between the *delegator* and *delegatee.* Also, though broadcasting the *delegation request* for all the peers is one option, it could be bandwidth intensive. Therefore, how to select most appropriate *candidate* peer remains as an issue in trust computing for future works.

References

1. Byeong-Ho, K.: Ubiquitous computing environment threats and defensive measures. Int. J. Multimedia Ubiquit. Eng. **2**(1), 47–60 (2007)
2. Albrecht, S.: Implicit human computer interaction through context. Pers. Technol. **4**(2–3), 191–199 (2000)
3. Meixner, G.: Past, present, and future of model-based user interface development. i-com **10** (3), 2–11 (2011)
4. Vanderdonckt, J.: Distributed user interfaces: how to distribute user interfaces elements across users, platform and environments. In: The 11th Congreso Internacional de Interacción Persona–Ordenador (Interacción 2010), Valencia, Spain, pp. 3–14 (2010)

5. Melchior, J., Grolaux, D., Vanderdonckt, J., Van Roy, P.: A toolkit for peer-to-peer distributed user interfaces: concepts, implementation and applications. In: EICS, USA, pp. 69–78 (2009)
6. Nichols, J., Myers, B.: Creating a lightweight user interface description language: an overview and analysis of the personal universal controller project. ACM Trans. Comput.-Hum. Interact. **16**(4), 17–37 (2009)
7. Radu, V.: Application. In: Radu, V. (ed.) Stochastic Modeling of Thermal Fatigue Crack Growth. ACM, vol. 1, pp. 63–70. Springer, Heidelberg (2015)
8. Nilsson, E., Floch, J., Hallsteinsen, S., Stav, E.: Model-based user interface adaptation. Comput. Graph. **30**(5), 692–701 (2006)
9. Ghiani, G., Paternò, F., Santoro, C.: On-demand cross-device interface components migration. In: Proceedings of the 12th international Conference on Human Computer Interaction with Mobile Devices and Services, MobileHCI 2010, Lisboa, Portugal, pp. 299–308 (2010)
10. Berti, S., Paternó, F., Santoro, C.: A taxonomy for migratory user interfaces. In: Gilroy, S. W., Harrison, M.D. (eds.) DSV-IS 2005. LNCS, vol. 3941, pp. 149–160. Springer, Heidelberg (2006)
11. Sørensen, H., Raptis, D., Kjeldskov, J., Skov, M.: The 4C framework: principles of interaction in digital ecosystems. In: ACM International Conference on Pervasive and Ubiquitous Computing (UbiComp 2014), USA (2014)
12. Elmqvist, N.: Distributed user interfaces: state of the art. In: Gallud, J.A., Tesoriero, R., Penichet, V.M.R. (eds.) Distributed User Interface. Human-Computer Interaction Series, pp. 7–12. Springer, London (2011)
13. Frosini, L., Paternò, F.: User interface distribution in multi-device and multi-user environments with dynamically migrating engines. In: EICS, USA, pp. 55–64 (2009)
14. Paternò, F., Santoro, C.: A logical framework for multi-device user interfaces. In: The 4th ACM SIGCHI Symposium on Engineering Interactive Computing Systems (EICS 2012), Denmark, pp. 45–50 (2012)
15. Sauter, P., Vogler, G., Specht, G., Flor, T.: A model–view–controller extension for pervasive multi-client user interfaces. Pers. Ubiquit. Comput. **9**(2), 100–107 (2005)
16. Grolaux, D., Van Roy, P., Vanderdonckt, J.: Migratable user interfaces: beyond migratory interfaces. In: The First Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services, MOBIQUITOUS 2004, Cambridge, pp. 422–430 (2004)
17. Pham, A.: Privilege delegation and revocation for distributed pervasive computing environments. In Abraham, G., Rubinstein, B., (eds.) Proceedings of the Second Australian Undergraduate Students' Computing Conference, pp. 136–141 (2004)
18. Haddadi, A.: Communication and Cooperation In Agent Systems: A Pragmatic Theory. Springer-Verlag, New York (1996)
19. Castelfranchi, C.: Towards a theory of delegation for agent-based systems. Robot. Auton. Syst. **24**(3), 141–157 (1998)
20. Mayhew, D.: The Usability Engineering Lifecycle: A Practitioner's Handbook for User Interface Design. Morgan Kaufmann Publishers, San Francisco, USA (1999)
21. Bendor, J., Glazer, A., Hammond, T.: Theories of delegation. Annu. Rev. Polit. Sci. **4**, 235–269 (2001)