# Universal Interactions with Smart Spaces

*Choonhwa Lee, Sumi Helal, and Wonjun Lee*

A critical challenge facing the pervasive computing research community is the need to manage complex interactions among numerous interconnected computers and devices. In such a pervasive space, a given application's functionalities are partitioned and distributed across several computing devices that are spontaneously discovered and used. In particular, because various devices will need to use the application's user interface, the interface must support—and be able to adapt to—various interaction modalities, device capabilities, and local computing resources.

In recent years, researchers have devoted much attention to universal interactions with diverse devices in richly networked settings. We can categorize the numerous approaches explored into two groups: *universal user interface languages* and *user interface remoting*. Using the universal-UI-languages approach, developers write the user interaction in an abstract language without targeting any particular device, so it can later be instantiated and presented to any given device. More specifically, a universal language describes user interfaces that are rendered by mapping a description's device-independent interaction elements to a target platform's concrete interface objects. The UI-remoting approach stems from service discovery frameworks and enables device interoperability using an agreed-upon user interface presentation protocol to remote devices.

Here, we review recent noteworthy efforts for universal interactions using these two approaches. Such efforts aim to raise interoperability in interactive smart spaces by standardizing user interface languages or communication protocols.

## UNIVERSAL INTERACTION TECHNOLOGIES

R&D efforts to facilitate networked-device interactions can be related to Web accessibility, abstract user interface description, and dynamic service discovery. The proliferation of diverse devices in recent years, together with the explosive adoption of Web portals, has created a need for increased Web accessibility—from any device, by anyone, at any location.

Device Independence (DI), an integral part of the World Wide Web Consortium's (W3C) efforts for unconstrained Web access, defines a framework of delivery context and adaptation. The delivery context conveys a client device's characteristics, and a server adapts the pages being accessed, modifying their layout and style to cater to the client's device capability and personal preference. The integrative framework lets us employ constituent component technologies.

For instance, the W3C Composite Capabilities/Preferences Profile (CC/PP) is the first attempt to define a vocabulary to describe delivery context. Also, the W3C has standardized its XForms—a platform-independent markup language for the next-generation Web form—for device-independent presentation purposes.

In addition to DI efforts, other numerous user interface languages exist, including

- the International Committee for Information Technology Standards Universal Remote Console (INCITS/V2 URC),
- the User Interface Markup Language (UIML),
- the Extensible Interface Markup Language (XIML), and
- Carnegie Mellon University's Personal Universal Controller (PUC).

These languages enable device-independent presentation by letting target devices determine the most suitable presentation from a given universal description in terms of a predefined set of abstract user interface components. The targets might be common IT devices, assistive-technology devices for the physically challenged, or other resource-constrained devices, which require support for various modalities such as visual, auditory, and tactile interfaces. For example, we should be able to present a text component in an abstract description as displayed text on computer screens, spoken audio for the blind, or output on Braille strips.

UI remoting takes a different approach from the universal UI languages. It uses a remote user interface protocol that lets an application interact with its user interface proxy exported to a remote device. The protocol relays I/O events between an application and its user interface, which resides on a remote machine. In this approach, a broader range of devices can control the application—even minimal-resource devices that can barely afford the remote user interface protocol.

UI remoting makes even more sense when used within a dynamic-service-discovery framework. This is because information about remote user interface protocols supported by applications and client devices and their capabilities can be used for lookup and matchmaking between both ends. The Universal Plug and Play (UPnP) Remote User Interface (RUI) standard and the μJini Proxy architecture (discussed later) belong in this category.

Various approaches to the universal-interaction problem can also be classified into different authoring styles, depending on whether the user interface targets specific client device platforms. The W3C DI articulates three possible cases: *single*, *flexible*, or *multiple* authoring. Single authoring automatically adapts a single generic description to different device capabilities (one size fits all). In contrast, multiple authoring develops a user interface for each type of client devices (custom made). This might offer the most complete user interface and best user experience but at an almost prohibitive development cost. Flexible authoring with a limited set of special user interfaces represents a compromise: customized user interfaces for popular platforms and automatically generated interfaces for rare platforms.

## UNIVERSAL UI LANGUAGES

The two front-runner universal UI languages are W3C XForms and INCITS/V2 URC. However, we also briefly compare UIML, XIML, and CMU PUC.

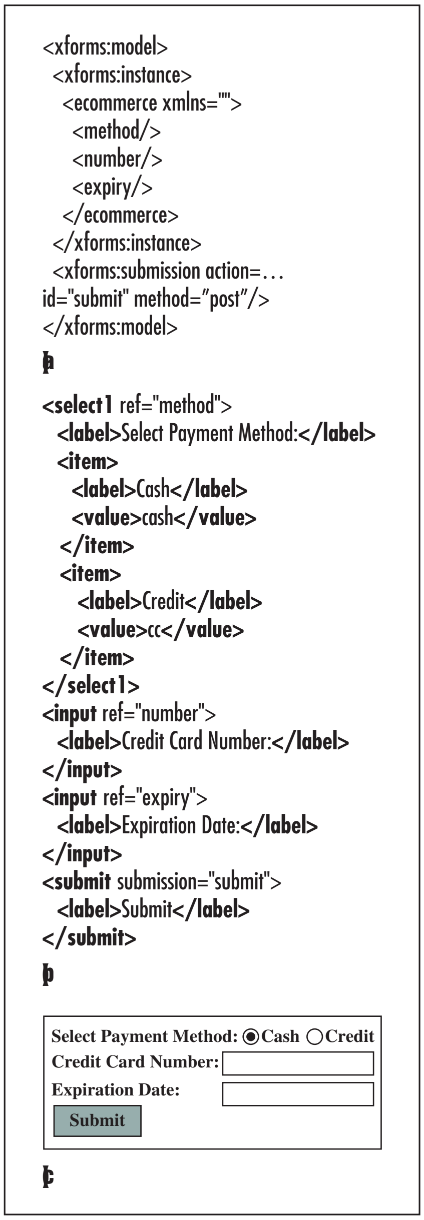
### W3C XForms

W3C XForms differs from HTML forms in that it separates content from presentation.[1] XForms documents consist of two sections: a data model (the XForms model) and data presentation (the XForms user interface and other presentation options). The XForms model is a template of an XML data instance being collected, and the data presentation describes how to display the data. An XForms user interface comprises a predefined set of generic interface elements, called XForms form controls, which capture a high-level logic of user interactions. Each maps to concrete interaction components on target access devices.

Figure 1 shows a sample e-commerce form in XForms:[1] an XForms model, its abstract user interface description, and a possible presentation on target devices. The <xforms:instance> element defines an XML template for data to be collected, while <xforms:submission> submits the collected data to the server. XForms form control elements are shown in bold in the abstract description. The <label> element might be displayed on PDAs or be spoken for the blind. The <select1> element might be rendered as a radio button or spoken options, and the <input> element inputs data. The example also shows that the form controls are bound to XML elements in their corresponding XForms model using the ref binding mechanism. For example, the <select1> element's ref attribute points to the <method> element in the XForms model, defining a link between the two sections of the data model and user interface.

### INCITS/V2 URC standards

INCITS/V2 URC is a set of standards enabling remote and alternative interfaces for information and electronic products.[2] The standards define a generic framework and an XML-based user interface language to let a wide variety of devices act as a remote to control other devices—called *Targets*. The January–March 2004 installment of this column, "The Universal Remote Console: A Universal Access Bus for Pervasive Computing," provided an architectural



```
<xforms:model>
  <xforms:instance>
    <ecommerce xmlns="">
      <method/>
      <number/>
      <expiry/>
    </ecommerce>
  </xforms:instance>
  <xforms:submission action=…
id="submit" method="post"/>
</xforms:model>
```

(a)

```
<select1 ref="method">
  <label>Select Payment Method:</label>
  <item>
    <label>Cash</label>
    <value>cash</value>
  </item>
  <item>
    <label>Credit</label>
    <value>cc</value>
  </item>
</select1>
<input ref="number">
  <label>Credit Card Number:</label>
</input>
<input ref="expiry">
  <label>Expiration Date:</label>
</input>
<submit submission="submit">
  <label>Submit</label>
</submit>
```

(b)

Select Payment Method: ● Cash ○ Credit
Credit Card Number:
Expiration Date:
Submit

(c)

Figure 1. A sample W3C XForms user interface: (a) a sample e-commerce XForms model, (b) its abstract user interface description, and (c) a possible user interface for target devices.[1]

overview of INCITS/V2 URC. Here, we focus on a sample user interface for a quick, comparative assessment with other approaches.

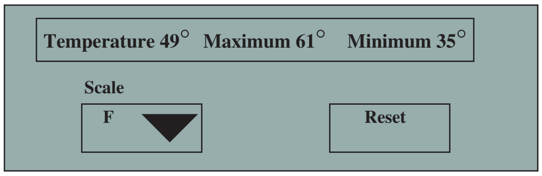
The URC approach models each Target's functional units as *User Interface Sockets*, which act as an access and control point to the Target. Consider a digital thermometer Target that displays the

```
<uiSocket about="http://www.mycorp.com/thermometer/socket"
      id="socket" xmlns="http://www.incits.org/incits390-2005"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <variable id="temperature" type="xsd:double">
        <dependency write="false()"/>
   </variable>
   <variable id="maximum" type="xsd:double">
        <dependency write="false()"/>
   </variable>
   <variable id="minimum" type="xsd:double">
        <dependency write="false()"/>
   </variable>
   <variable id="scale" type="scaleType"/>
   <command id="reset"/>
   <notify id="checkReset" explicitAck="false" category="alert">
        <dependency acknowledge="value('confirmReset')='done' or value('cancelReset')='done'"/>
   </notify>
   <command id="confirmReset" type="uiSocket:basicCommand">
        <dependency read="value('checkReset')='active'" execute="value('checkReset')='active'"/>
   </command>
   <command id="cancelReset" type="uiSocket:basicCommand">
        <dependency read="value('checkReset')='active'" execute="value('checkReset')='active'"/>
   </command>
   <xsd:schema>
      <xsd:simpleType name="scaletype" id="idScaleType">
          <xsd:restriction base="xsd:string">
          <xsd:enumeration value="F"/>
          <xsd:enumeration value="C"/>
        </xsd:restriction>
      </xsd:simpleType>
   </xsd:schema>
</uiSocket>
```

(a)

```
<pret name="http://www.mycorp.com/thermometer/corepret"
   id="pret"
     xmlns="http://www.incits.org/incits391-2005">
   <dcterms:conformsTo>http://www.incits.org/incits391-2005</dcterms:conformsTo>
   <group id="readings">
      <output id="temperature" ref="http://www.mycorp.com/thermometer/socket#temperature"/>
      <output id="maximum" ref="http://www.mycorp.com/thermometer/socket#maximum"/>
      <output id="minimum" ref="http://www.mycorp.com/thermometer/socket#minimum"/>
   </group>
   <select1 id="scale" ref="http://www.mycorp.com/thermometer/socket#scale"/>
   <trigger id="reset" ref="http://www.mycorp.com/thermometer/socket#reset"/>
   <modalDialog id="checkReset" ref="http://www.mycorp.com/thermometer/socket#checkReset">
      <trigger id="confirmReset" ref="http://www.mycorp.com/thermometer/socket#confirmReset"/>
      <trigger id="cancelReset" ref="http://www.mycorp.com/thermometer/socket#cancelReset"/>
   </modalDialog>
</pret>
```

(b)



(c)

current temperature, recent minimum and maximum temperatures, and the scale either in Fahrenheit or centigrade.[2] It also lets users reset the minimum and maximum temperature to the current temperature. A UI Socket description in figure 2 describes the Target's state and functionality using Variables, Commands, and Notifications. The Variables are state variables to indicate dynamically changing information of the Target—the current temperature in figure 2. URC invokes the Commands to ask the Target to perform certain functions such as a "reset." The Target triggers Notifications to notify URC users of certain events. In figure 2b, checkReset checks on a reset request with its users.

A modality-independent user interface specification—the presentation template in figure 2—is accompanied with the abstract-functionality description. It provides the socket presentation information by describing a structure of abstract interactors, each of which binds to the socket elements. This is much like mapping XForms form controls to a data model element. Moreover, the interactors are largely a subset of XForms 1.0 form controls, with a few exceptions. The presentation template includes the output interactor for displaying a value of variables and commands, select1 for a single choice, trigger for triggering a command bound to a command element in the socket description, and modalDialog for a target-triggered dialog (which is bound to a notify element in the socket description). Note that the ref attribute in each interactor establishes a binding to relevant socket elements. Besides, the group element organizes individual interactors in a hierarchical fashion. Figure 2c shows a possible user interface resulting from the socket description and presentation template.

**Figure 2. A sample INCITS/V2 Universal Remote Console user interface: (a) a digital thermometer UI Socket description, (b) the corresponding presentation template, and (c) a possible user interface.[2]**

### UIML, XIML, and PUC

Sharing the same philosophy separating an abstract interface description and its later rendering in any delivery context, UIML, XIML, and CMU's PUC define a set of basic interaction elements, a mechanism for grouping such elements, and optional additional presentation information.

A UIML document has styling sections that map interface elements to target UI objects (such as GUI widget classes), implying the need for one style section for each target device type. However, unlike other user interface languages, UIML doesn't support an explicitly separate data model, resulting in undesirable data fusion in the user interface elements.

An XIML presentation component defines concrete interaction elements for a particular target platform. Similar to UIML, it can support a new device type by defining one presentation component, meaning it follows the multiple authoring approach. Alternatively, a single intermediate presentation component can describe XIML interfaces and automatically create a concrete presentation component using a set of predefined relations. Therefore, XIML (like INCITS/V2 URC) belongs to the flexible authoring style that allows multiple implementations, each fine-tuned to a particular device class, apart from the all-in-one abstract description.

PUC describes device functions in terms of state variables and commands reminiscent of those of INCITS/V2 URC. In addition to its grouping mechanism as a hint for placing relevant components closer together, it can specify dependency information to indicate whether a component is *activatable* with regard to others. So, it can gray out or remove unusable parts on a small-display device. This feature is somewhat similar to the URC dependency element of figure 2.

## UI REMOTING

With UI remoting, user interfaces reside on a remote platform instead of the one running their applications. This lets a multitude of wired and wireless devices be used as I/O devices. A remote user interface protocol conveys I/O events of key inputs and display updates between user interfaces and applications.

For example, a home security system residing on a set-top box could rely on various kinds of devices to signal an alarm, provided they speak the same remote user interface protocol. A TV screen in the living room or a touchpad attached to the kitchen refrigerator might display an alarm, depending on a resident's location. The devices could also relay to the security application user responses such as detailed alarm information retrieval and subsequent deactivation.

We could further improve UI remoting's effectiveness by incorporating it into a service discovery framework. The UPnP RUI standards and μJini Proxy Architecture have recently explored such a marriage, because it offers more options for solving the user interface presentation problem. First, the device-independent user interface languages we discussed earlier can be instantly used. An abstract service user interface description might be made available to clients as part of service advertisements for a concrete-interface generation, which belongs to the single authoring case. Second, multiple device-specific user interface implementations can be attached to service advertisements, so that clients can select the most appropriate one to their device capability and preference. Therefore, current service discovery frameworks support the flexible authoring style (without additional efforts).

### UPnP RUI

The UPnP RUI standard, first published in September 2004, added UPnP RemoteUIClientService and RemoteUIServerService on top of the basic UPnP device architecture. The standard defines UPnP supports, which map compatible applications and remote UIs and establish, maintain, and terminate a control session between them. The RemoteUIServerService is a UPnP service that exposes a list of compatible UIs, whereas a RemoteUI client device displays the user interfaces. The matchmaking is based primarily on supported remote I/O protocols on both sides, including AT&T Virtual Network Computing (VNC), the Microsoft Remote Desktop Protocol (RDP), Intel Extended Remote Technology (XRT), and the Internet Engineering Task Force's Lightweight Remote Display Protocol (LRDP).

The UPnP RUI standard supports two usage scenarios. In the first scenario, UPnP RUI control point connects an RUI client to remote applications. A UPnP RUI server exposes a list of remote-able user interfaces. An RUI server lets an RUI control point on the network browse interfaces by calling the server's GetCompatibleUIs action to retrieve a list of interfaces compatible with a target client device. The control point invokes the Connection action on the client, connecting an out-of-band remote I/O to a designated remote-able application. In other words, the connection setup results in a remote UI executing on the client device.

In the second usage model, available user interface lists are directly pushed to a UPnP RUI client, so that it can initiate a connection by itself. A user chooses one of the user interfaces to view and control, using a local user interface selection menu presented on the client device. For this, the RUI client stores the list of compatible user interfaces, which RUI client control points on the network manage using AddUIListing, GetUIListing, and RemoveUIListing action invocations.

### μJini Proxy Architecture

Researchers have prototyped and demonstrated a Jini equivalent to the UPnP RUI standard on mobile phones.[3] Jini has been extended into a μJini Proxy Architecture that supports context-aware service discovery as well as UI remoting. Targeting interactive spaces inhabited by a range of diverse devices, the system functions as a proxy for resource-con-

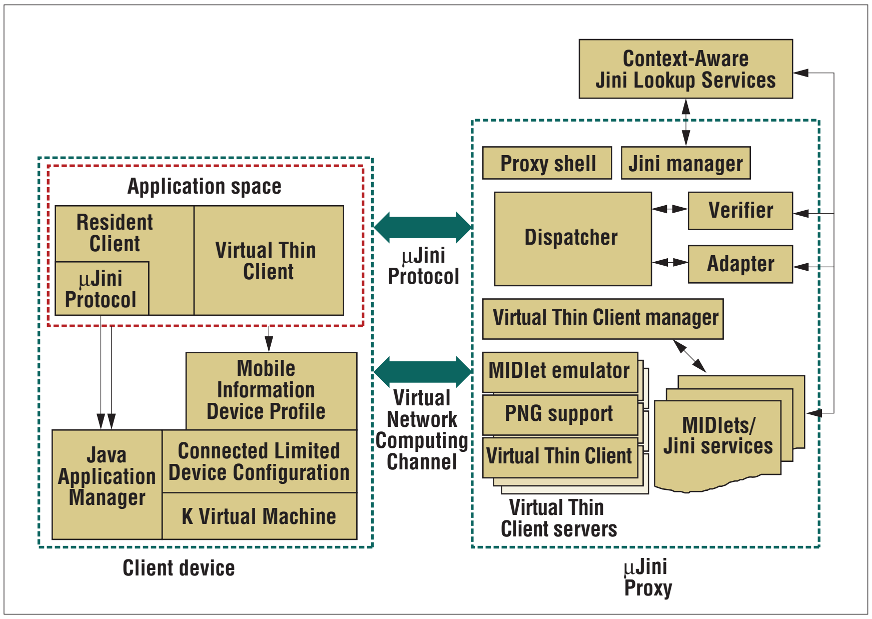


**Figure 3.** μJini Proxy system architecture.

strained devices that can't afford to engage in the Jini protocol. The prototype used an AT&T VNC protocol as a remote I/O protocol between client devices and services residing on the proxy side. The Jini service matching mechanism has also been extended to include dynamically changing properties relevant to services and network condition, as well as client devices.

The μJini Proxy Architecture comprises four main components: μJini Protocol, Resident Client, Virtual Thin Client (VTC), and μJini Proxy (see figure 3). The Resident Client resides on a J2ME client device and takes care of interfacing between the client and the proxy server as well as handling remote UI messages through a VNC channel. The μJini Proxy performs context-aware service discovery on the network side for the client and executes a thin-client server for UI-remoting a discovered service. The VTC is split into two pieces, VTC Client on the client side and VTC Server on the μJini Proxy side. The μJini Protocol lets the client and proxy exchange session management requests and responses, and a separate VNC channel conveys remote I/O events.

It's important to know that the two approaches—universal UI languages and UI remoting—are orthogonal (see table 1 for a comparison of the two). The former doesn't concern itself with service discovery issues, simply assuming an underlying service discovery platform. The latter might be used in parallel with device-independent user interface languages over a service discovery framework, producing synergetic effects.

Furthermore, tackling the same problem naturally leads to overlapping technologies, built on others' success. For instance, INCITS/V2 URC seems closer to UPnP than to others in that it first examines the core service information, then retrieves further details. The two

TABLE 1
Comparison of universal UI language and UI remoting approaches.

| | Universal UI language approach | | UI remoting approach | |
|---|---|---|---|---|
| | **W3C XForms** | **INCITS/V2 URC** | **UPnP Remote UI** | **μJini Proxy Architecture** |
| **Origin** | Unified Web access | Universal Remote Console | Spontaneous networking | Spontaneous networking |
| **Functionality modeling** | XForms data model | UI Socket | UPnP service | Java service interface |
| **Description form** | XML/XPath (XForms user interface) | XML/RDF (Presentation Template) | XML/UPnP | Java class |
| **Client and server** | Web browser and Web server | URC and Target | UPnP RUIClientService and RUIServerService | Virtual Think Client and Java service |
| **Discovery** | External means | Target description and underlying discovery framework | UPnP RUIClient/Server and UPnP Simple Service Discovery Protocol | Enhanced context-aware service discovery |
| **Delivery** | Serialized representation | Underlying networking platform | Remote user interface protocols | Remote user interface protocols |
| **User interface description component** | XForms form control | URC interactors | Remote user interface messages | Remote user interface messages |

also share the concept of state variables and commands. Another example is that INCITS borrowed most of its URC interactors from W3C XForms.

These intertwining evolutions are pushing the envelope toward a truly universal interaction technology for pervasive spaces.

## ACKNOWLEDGMENTS

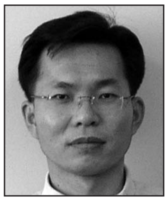
Grant R01-2005-000-10267-0 from the Korea Science and Engineering Foundation in the Ministry of Science & Technology partly supported this work.

## REFERENCES

1. M. Dubinko et al., *XForms 1.0*, World Wide Web Consortium (W3C) recommendation, Oct. 2003; www.w3.org/TR/2003/REC-xforms-20031014.

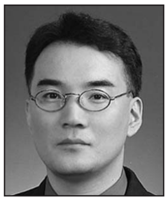
2. INCITS/V2, *ANSI INCITS 389-2005: Protocol to Facilitate Operation of Information and Electronic Products through Remote and Alternative Interfaces and Intelligent Agents: Universal Remote Console*, Feb. 2005, www.ncits.org/list_INCITS.htm.

3. C. Lee, A. Helal, and D. Nordstedt, "μJini Proxy Architecture for Impromptu Mobile Service Access," *Proc. Workshop Next Generation Service Platforms for Future Mobile Systems* (SPMS 06), 2006.



**Choonhwa Lee** is an assistant professor in the College of Information and Communications, Hanyang University. Contact him at lee@hanyang.ac.kr.

**Sumi Helal** is a professor at the University of Florida and is the director of its Mobile and Pervasive Computing Laboratory. He is also president and CEO of Phoneomena. Contact him at helal@cise.ufl.edu.



**Wonjun Lee** is an associate professor in the Department of Computer Science and Engineering, Korea University. Contact him at wlee@korea.ac.kr.