2016 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)

The Transreality Interaction Platform: Enabling Interaction Across Physical and Virtual Reality

Kyle A. Martin
Dept. of Computer Science
University of Central Florida
Orlando, Florida
Email: kmartin@cs.ucf.edu

Joseph J. LaViola Jr.
Dept. of Computer Science
University of Central Florida
Orlando, Florida
Email: jjl@cs.ucf.edu

***Abstract*—The convergence of the Internet of Things (IoT) and interactive systems will enable future interactive environments which transcend physical and virtual reality. *Embedded Things* provide sensors and actuators to virtualize the physical environment, while *Interactive Things* extend the virtualized environment with modalities for human interaction, ranging from tangible and wearable interfaces to immersive virtual and augmented reality interfaces. We introduce the Transreality Interaction Platform (TrIP) to enable service ecosystems which situate virtual objects alongside virtualized physical objects and allow for novel ad-hoc interactions between humans, virtual, and physical objects in a *transreality environment*. TrIP provides a generalized middleware platform addressing the unique challenges that arise in complex transreality systems which have yet to be fully explored in current IoT or HCI research. We describe the system architecture, data model, and query language for the platform and present a proof-of-concept implementation. We evaluate the performance of the implementation and demonstrate its use integrating embedded and interactive things for seamless interaction across physical and virtual realities.**

I. Introduction

The convergence of the Internet of Things (IoT) and interactive systems will have profound effects on the design and implementation of future interactive environments (IEs). While some research in the Human Computer Interaction (HCI) and IoT communities has begun to explore these effects, the potential for overcoming the boundaries between physical and virtual reality has yet to be fully explored. The bulk of previous IoT research has focused on non-interactive *Embedded Things*, which are hidden in the environment and provide sensors or actuators which collectively virtualize our physical environment to enable new kinds of interactions between humans, services, and the physical world. We introduce the concept of *Interactive Things* to refer to the combination of an emerging subset of IoT devices providing human interface modalities and networked Human Interface Devices (HIDs) not traditionally considered as IoT devices. Extending the virtualized environment (VE) provided by Embedded Things with the interface modalities of Interactive Things enables *transreality* interaction across physical and virtual environments. Service ecosystems leverage *transreality environments (TEs)* to situate virtual objects alongside virtualized physical objects to achieve novel ad-hoc interactions amongst humans, virtual, and physical objects.

Transreality systems are highly complex and present a number of unique challenges which have yet to be addressed in HCI or IoT literature. Chief amongst these is decoupling services from user interface (UI) implementations to facilitate interaction with users across the wide variety of interface modalities present in a transreality environment. Ensuring responsiveness while protecting user privacy and security additionally requires a novel architecture departing from the traditional multi-tier and big data architectures commonly used for implementing UIs and IoT systems. While some previous work has studied the intersection of IoT and interactive systems, such as the use of IoT devices in Augmented Reality (AR) applications, no known work has explored these specific challenges.

We introduce the Transreality Interaction Platform (TrIP) to address these challenges by providing a novel general-purpose middleware implementation of the Distributed View-Model (DVM) design pattern previously demonstrated for Distributed User Interfaces (DUIs). A DVM is a network shared object which represents the state and functionality of a UI element bound to a portion of an application domain model, effectively decoupling the application domain logic (Model) from the implementation of the application UI (View). TrIP is a specialized event-driven object database for managing DVMs represented as live XML DOM objects with out-of-band metadata for indexing, security policy, and other ViewModel properties. It provides a query API for defining, manipulating, and brokering events on DVMs, which is used by backend services managing domain Models and reusable frontend clients providing physical and virtual Views.

In this paper, we describe the requirements of transreality systems and detail the architecture, object model, and query language for TrIP. We evaluate the performance of a proof-of-concept TrIP implementation and demonstrate its use in a simple transreality environment scenario. Results show the present implementation provides an adequate foundation for exploration of transreality interactions and also indicate areas for future transreality systems research and development.

II. Related Work

The Internet of Things is expected to be composed of more than 38 billion devices by 2020 [1]. The majority of these devices are Embedded Things with sensors and actuators that collectively virtualize an environment and the physical objects contained therein, e.g., RF or visual object tracking and spatial mapping sensors or control systems providing remote control of equipment. Up to 12 billion IoT devices are expected to be consumer oriented, and many of those will be interactive, such as wearables, tangibles, domotics, and smart space devices [2], [3]. *Interactive Things* are distinguished from Embedded Things by providing human interface modalities that can be discovered and appropriated by ecosystem services for situated user interaction [4], [5].

Much of present IoT research is derived from earlier research on RFID sensor networks and sensor databases [6]–[10]. This lineage has led to a strong focus on sensor data management and big data solutions for supporting the extremely large volume of sensor data expected to be generated by billions of IoT devices [11]. To a lesser extent, there has also been research on exposing service-oriented interfaces for controlling large numbers of IoT devices individually and in aggregate [6], [7]. A common thread in both is sharing sensor data and device controls at scale to foster multi-application ecosystems [11]. One significant shortcoming of current IoT platforms is the increased scale and centralization in the cloud reduces response times, especially when services are not collocated with the platform and require another round trip to handle an interaction. Lack of end-user control of data has also been cited as a shortcoming of current IoT platforms and enforcing end-user control has led to federated IoT platforms being proposed as a privacy and security preserving alternative to cloud-based platforms [7], [12]. While IEs have been frequently mentioned as application areas in IoT literature, there is comparatively little corresponding research in HCI literature, and no literature found from either addressing the challenges and requirements for supporting IoT devices within IEs [6], [7], [13].

The virtuality continuum classifies perceived reality along a spectrum ranging from purely physical reality, to mixed and augmented reality (MR and AR), and finally to purely virtual reality (VR) [14]. Each point along the spectrum is associated with specific interface modalities with varying degrees of mediation between user perception and their physical reality [15]. Examples include tangible interfaces for physical reality, mobile phones for augmented reality, and Head Worn Displays (HWDs) for virtual reality. Previous work in this area has been limited to studying interaction at specific points along the spectrum. The only known work to consider simultaneous interaction across multiple points along the spectrum comes from research on pervasive gaming which introduced the concept of *transreality gaming* [16], [17]. We have generalized and significantly extended these original concepts to incorporate IoT devices for everyday interaction across physical and virtual realities.

One of the few examples of HCI research on IoT presented several examples using IoT devices as Human Interface Devices (HIDs) and derived guidelines for embedded interaction with IoT [18]. A number of papers describe AR interaction with IoT devices, such as using visual tracking to facilitate AR overlays for IoT devices [19]–[24] or sensor based object tracking to provide augmented virtuality for tangible UIs (TUIs) in toys, gaming, and entertainment applications [25]–[30]. The concept of mirror worlds considers mixed reality environments where situated agents can augment the physical world through a VE consisting of virtual objects coupled to IoT devices [31]–[34]. While mirror worlds share a number of similarities with transreality, research has been limited to interactions between agents and the VE and does not consider the much broader space of interactions possible between humans, virtual, and physical objects.

One of the primary challenges to implementing complex interactive systems is reducing the coupling between application logic and UI implementation to avoid the costs of developing platform specific UIs. Cross-platform UI toolkits and UI design patterns like Model-View-Control (MVC), Model-View-ViewModel (MVVM), and Model-View-Presenter (MVP) help to manage the complexity of UI design and implementation and enforce decoupling between domain and UI logic [35]–[37]. However, even when using these techniques, supporting a wide range of device platforms still requires the development of platform specific implementations. Distributed User Interfaces (DUIs) are complex interactive systems that allow application UIs to be distributed across multiple devices, such as tabletop, wall mounted, and mobile screens, necessitating complex state synchronization and event routing between hosts [38]. Research on DUIs has yielded the most promising approach to managing complex multi-platform interactive systems by adapting UI design patterns to distributed systems.

The Distributed ViewModel (DVM) pattern, which derives from the MVVM pattern, effectively decouples application logic from UI implementation [39], [40]. It consists of network-shared ViewModel objects stored in an object database and synchronized across clients implementing application logic and platform specific Views. A similar approach derives from the MVP pattern to provide a distributed multimodal UI framework consisting of Views on hosts providing interface modalities, shared synchronized ViewModels, and Presenters running on hosts managing the Model and domain logic for an application [41]. Ravel is a data-flow oriented IoT framework that generates code for distributing MVC components across IoT sensor devices, gateways, and cloud hosts while coordinating components based synchronizing portions of shared Model state across the hosts [42]. Utilizing an object database to implement the DVM rather than a shared ViewModel runtime provided by a framework [41], [42] significantly decouples the ViewModel from View and Model component implementations. This approach frees developers to implement Views and Models using any platform supported by the object database client, rather than the constrained set of platforms supported by the runtime. The shared ViewModel also significantly reduces the complexity of implementing DUIs, where a shared ViewModel can be easily synchronized and adapted across different hosts by binding to Views suitable for each host's available interface modalities.

III. Transreality Systems

Transreality environments (TEs) consist of two types of devices: Embedded Things and Interactive Things. Embedded Things are non-interactive IoT devices with sensors and actuators which collectively virtualize the physical environment. These may be infrastructural devices, such as building or environmental control systems, or scanning and tracking devices which monitor physical objects in the environment. While not directly interactable, they provide additional affordances through the network to facilitate indirect interaction with devices which would otherwise not be possible, such as remote control and configuration interfaces. They also enable interaction through non-networked physical objects sensed in the environment, such as using RFID tags placed on objects or visual object recognition and tracking. Interactive Things are IoT devices which extend the VE with modalities for human interaction. These include everything from tangible and wearable IoT interface devices like buttons, medical sensors, and watches, to traditional human interface devices (HIDs) like mounted and mobile screens, to immersive AR and VR HWDs.

The virtualized environment and interface modalities provided by Embedded and Interactive Things combine to form a seamless transreality environment enabling interaction across a wide range of modalities. There are three types of objects that may be situated within a TE. *Virtualized* objects provide 3D representations of Embedded or Interactive Things which mimic the appearance and interactions of the physical objects using available modalities. Examples include a thermostat providing temperature control for a particular room in a smart building, or smart appliances in a kitchen workspace, or spatially tracked tangible controllers like those used with the HTC Vive. *Virtual* objects are 3D objects with appearance and interactions which imply physicality. Interaction with virtual objects is similar to virtualized objects with the exception of not being coupled to a physical device. Examples of virtual objects include 3D avatars for agents, 3DUIs for services, or arbitrary 3D content situated in the environment, like virtual sculptures or flowers. *Meta* objects are 2D or 3D objects with no implied physicality, such as abstract 3DUIs, GUIs, documents, and multimedia. These objects are most commonly encountered using traditional screen-based HIDs and enabling placement within transreality environments opens up new possibilities for novel interactions and collaboration.

Service ecosystems leverage the transreality environment to situate virtual and meta objects alongside virtualized physical objects to enable novel interactions that have not been possible in previous IEs. Transreality objects within the environment may be situated within the VE, or may be mapped from remote environments into the VE around a user. Any number of available modalities provided by the TE may be utilized for interaction with transreality objects and users may seamlessly utilize multiple modalities simultaneously or transition between modalities over time. Managing multi-modal interaction with both virtual objects and Embedded and Interactive Things, while simultaneously exposing the environment to service ecosystems for interaction, is an extremely challenging problem which no previous work in either HCI or IoT research is known to have considered. We introduce the Transreality Interaction Platform (TrIP) to address these challenges.

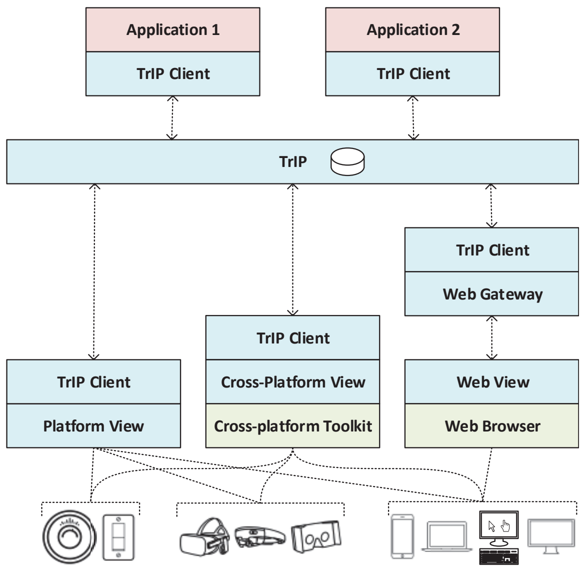

Fig. 1. The TrIP system architecture enables service ecosystem interaction across a wide range of Embedded and Interactive Things using general purpose clients implemented for specific platforms, using cross-platform toolkits, or web-based interfaces.

IV. The Transreality Interaction Platform

TrIP is a specialized event-driven object database designed to provide the first known general purpose implementation of the Distributed ViewModel design pattern to support development of transreality environments [39], [40]. TrIP manages DVMs and provides a query interface for defining and manipulating DVMs, synchronization mechanisms for sharing DVMs between multiple clients, and granular event brokerage over DVM XML data structures. DVMs are managed as persistent live XML DOM trees with out-of-band metadata used internally for indexing, security policies, and associating type-specific properties. TrIP places no restrictions on the schema of ViewModels in order to allow the many available XML UI languages [43] and emerging IoT modeling languages [44], [45] to evolve over time. Reusable platform-specific TrIP clients adapt and bind DVMs to Views supporting available platform modalities, including the physical Views of Interactive and Embedded Things.

A. Architecture

Although Interactive Things provide human interface modalities that services can utilize for interaction, IoT devices are not traditionally treated as presentation components in a multi-tier architecture [46]. Furthermore, current IoT platforms expose devices through data management and service-oriented interfaces, which reside within the data and application layers of a multi-tier architecture. This disparity complicates development of transreality environments using existing IoT platforms as aspects of the presentation layer leak into the application and data tiers. TrIP addresses this issue by acting as a presentation layer middleware that provides a unified interface to both Embedded and Interactive Things available in a transreality environment.

The proposed architecture for transreality systems is shown in Fig. 1. Decoupling service logic from UI implementation through DVMs allows for a wide variety of Embedded and Interactive Things to be supported through reusable general purpose TrIP clients. These clients adapt ViewModels to available platform modalities and may be implemented using native platform interfaces, cross-platform toolkits, or web-based interfaces. Services utilize TrIP clients within their presentation layer to appropriate virtualized objects and situate virtual and meta objects within the TE to facilitate user interaction. Service ecosystems share the TE to interact with users and may also interact amongst themselves through situated objects, such as an agent represented by an avatar which can interact with services on behalf of a user through objects in the TE.

Unlike most Embedded Things, the latency requirements for Interactive Things are much more stringent to ensure responsiveness [47]. Interactive Things also expose significantly more personal information than non-interactive devices. These two factors favor a decentralized platform that can minimize latency and ensure end-user control of collected information [7]. Most IoT platforms, however, are centralized in the cloud and employ big data techniques to manage the large number of devices and volume of collected sensor data. A centralized platform incurs higher latency than a decentralized platform as there are at least two Internet hops between an IoT device and a service (unless the service is collocated on the centralized platform), compared to the single hop from a decentralized platform to the service outside of the local network. For these reasons, the present implementation of TrIP is architected as a standalone server connecting with Embedded and Interactive Things through the local network to provide local or cloud-based services access to the local transreality environment.

B. Distributed ViewModels

The Distributed ViewModel design pattern [39], [40] provides a powerful tool for developing complex interactive systems such as transreality environments. TrIP implements the pattern using XML DOM data structures as ViewModels. Compared to JSON and other serialization formats, XML is widely used for UI modeling languages [43] and a number of emerging IoT modeling languages [44], [45]. The XML DOM specification also includes mechanisms for dispatching events through the DOM tree [48] and generating events in response to mutations [49]. TrIP leverages these capabilities to synchronize DVMs shared across multiple clients and provide event brokerage over XML trees.

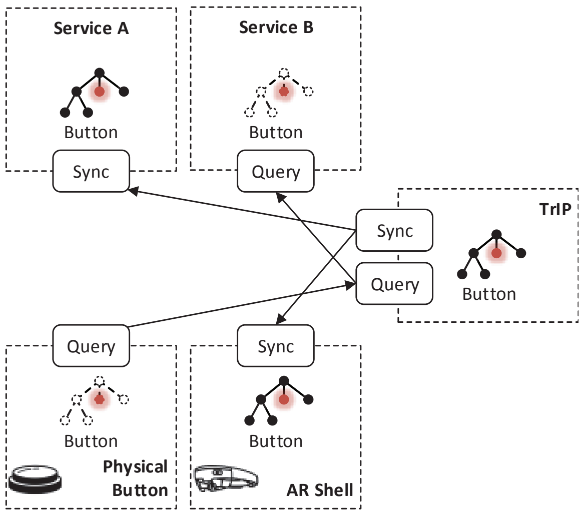

Fig. 2. An example of a DVM shared across multiple frontend and backend TrIP clients. A resource constrained *Physical Button* modifies a node in its ViewModel using a query. On the TrIP server, the live DOM of the button ViewModel is updated, triggering an event sent to XML node subscribers (*Service B*) and synchronized clients (*Service A* and the *AR Shell*).

When a client submits a query to modify a ViewModel, such as a resource limited IoT button as shown in Fig. 2, the modification is applied to the persistent ViewModel managed by TrIP, which in turn triggers mutation events sent to synchronized clients with local copies of the ViewModel (*Service A* and the *AR Shell*), or clients which have subscribed to mutation events on a particular XML node (*Service B*). Similarly, when a client modifies a local copy of a ViewModel, the mutation event is relayed to the TrIP server and applied to the persistent ViewModel, which in turn dispatches events to other synchronized clients or subscribers. The same mechanism is used for general events which may be published by dispatching the event on a node in a local copy of a ViewModel or by submitting a publish query specifying the node. Mutations and event dispatches must be “replayed” on the TrIP server to allow for subscribers to mutations or events on particular XML nodes to be properly notified. Synchronized ViewModels are not always feasible, especially for resource constrained devices with limited memory and processing capabilities insufficient for synchronizing an XML DOM structure. For this reason, TrIP provides a query API for performing equivalent modifications and event dispatching on ViewModels using XPath for selecting nodes and a set of modification commands for altering the XML state and structure.

While the use cases for DVMs are well studied for UIs, nothing similar is known to have been proposed for IoT devices. If Embedded and Interactive Things are considered as physical Views, then their ViewModels represent their state, functionality, and metadata — data typically managed in aggregate by current IoT platforms. Using these ViewModels, services can discover and appropriate devices based on contextual metadata to enable situated interaction within a TE.

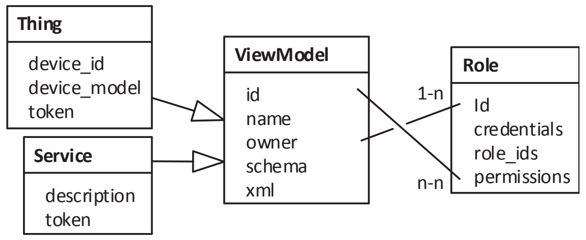

Fig. 3. TrIP object model

They can retrieve ViewModels to observe the state of device sensors or subscribe to events on the ViewModels to receive real-time sensor data. Likewise, manipulating the state of ViewModels or dispatching events on ViewModels can be used to alter the configuration of a device, invoke functionality, and control actuators. Appropriation involves a service requesting permissions from a device owner (if not already granted) to observe and manipulate a device's ViewModel. Depending on the device capabilities, multiple services may be given permissions on a single ViewModel, allowing sensor data to be shared among services or control of different modalities of a device to be assigned to specific services.

C. Object Model

The proposed object model for TrIP is shown in Fig. 3. The present design of TrIP provides a general-purpose *ViewModel* object intended for use with virtual and meta objects. Every *ViewModel* has a universally unique identifier, a name, and an owner assigned by security policy. Each also contains the XML DOM data structure which is serialized along with other assigned metatdata. An out-of-band schema property is assigned to each *ViewModel* to allow queries to select *ViewModel* by schema and support assigning schemas independent of any schema defined in the XML to facilitate evolution and richer typing. For virtualized objects like Embedded and Interactive Things, a *Thing* object, which descends from the *ViewModel* type, adds additional metadata such as device identification, model information, and security tokens. *Services* are another type of *ViewModel* which represents an ecosystem service and allows services available in a TE to be discoverable and interactable, such as for launching a service by publishing an event on its ViewModel.

Roles are security policy objects used to associate credentials and other metadata with users and for storing permission assignments. Every *ViewModel* is assigned an owner by reference to a *Role* ID. *Roles* can also represent policy groups or delegates by assigning one or more other *Roles* to form a hierarchy. Permissions assigned to a *Role* may be inherited by delegates and delegates may also be assigned ownership of ViewModels. Token-based authentication allows *Things* and *Services* to be authorized to assume an assigned role without supplying credentials. This also allows resource constrained devices to issue queries for updating state and publishing events in a single message, avoiding the typical handshake used during credential exchange.

```
QUERY :=  VIEWMODEL | VIRTUALIZED | ROLE
VIEWMODEL :=
  ( 'viewmodel' | 'thing' | 'service' ) '.'
  ( 'insert', DEFINITION+
  | 'find', SELECTOR, [PROJECTION]
  | 'update', SELECTOR, MODIFIER
  | 'destroy', SELECTOR
  | 'subscribe', SELECTOR, EVENT+
  | 'publish', SELECTOR, EVENT+, ARGUMENTS?
  | 'chown', SELECTOR, Role.id )
VIRTUALIZED :=
  ( 'thing' | 'service' ) '.'
  ( 'authorize', SELECTOR, Role.id?
  | 'status', SELECTOR
  | 'properties', SELECTOR, PROPERTIES? )
ROLE :=
  'role.'
  ( 'create', CREDENTIAL
  | 'add', Role.id, Role.id+
  | 'remove', Role.id, Role.id+
  | 'destroy', Role.id
  | 'credentials', Role.id, CREDENTIAL
  | 'grant', Role.id, PERMISSION+, SELECTOR
  | 'revoke', Role.id, PERMISSION+, SELECTOR )
DEFINITION :=
  JsonObject[ type properties ]
SELECTOR :=
  JsonObject[ type properties | XPath ]
MODIFIER :=
  JsonObject[ type properties | modifier command ]
EVENT := String
ARGUMENTS := JsonObject | JsonArray
PROPERTIES := JsonObject
CREDENTIAL :=
    JsonObject['username','password']
  | JsonObject['username','certificate']
  | JsonObject['token']
PERMISSION :=
    JsonObject['permission']
  | JsonObject['permission','object':SELECTOR]
```

Fig. 4. TrIP query syntax. Production rules determine a sequence of elements in a JSON array representing the query.

D. Query Language

In the present design of TrIP, a simple JSON protocol is used and a query language based on JSON arrays allows for efficient query parsing. The query syntax is summarized in Fig. 4. *ViewModels* and descendants support typical Create Retrieve Update Delete (CRUD) operations. When inserting new *ViewModels*, one or more JSON objects containing the serialized XML and associated metadata are provided. Finding, updating, and destroying *ViewModels* requires a selector JSON object containing values for a subset of the properties to match against. An XPath property may also be provided to retrieve *ViewModels* by matching against the XML DOM structure. Update queries also include a modifier JSON object, which may set property values or contain commands for performing XML DOM modifications, like setting or removing attributes and adding or removing child elements. Publish and subscribe queries on *ViewModels* can specify simple events on the *ViewModel* itself or on particular XML DOM nodes if an XPath is provided in the selector. Finally, ownership can be changed on one or more selected *ViewModels*.

Thing and *Service* objects support several additional queries. Authorization tokens may be assigned to allow devices or services to assume a particular *Role* by supplying the token. The network connectivity status of a *Thing* or *Service* may also be queried to determine if the device or service client associated with the object is connected. *Thing* and *Service* metadata properties, such as device identifiers or service descriptions may be retrieved or set.

Roles provide queries for basic CRUD operations, setting credentials for a particular *Role*, and granting and revoking permissions. The permissions assigned to a *Role* may be object-neutral for assigning broad capabilities to users, such as administrative rights, or may be specific to particular objects or object types.

E. Query Examples

Consider the ViewModel and queries (Fig. 5) used for an IoT button device providing a tangible UI in a simple transreality environment (Fig. 7). The button provides a sensor for detecting button presses and an actuator for controlling an RGB LED for indicating state. The XML ViewModel for the button provides a simple representation of the device (lines 1-5). The `button` element represents the button sensor and when the button is pressed it dispatches a `press` event on this element. The `led` element represents the LED actuator and the `state` attribute can be modified by a service to turn the LED on or off to notify a user. If the button device had sufficient capabilities, the firmware could use a synchronized ViewModel to allow it to directly alter the local XML DOM or listen and dispatch events on DOM nodes. In these examples we assume the button device has limited capabilities and must rely instead on the query API.

To create the ViewModel, the button issues an `insert` query with a JSON object containing the XML and metadata (lines 6-8). The result of this query will be a JSON object for the ViewModel, including any default metadata properties added during insertion, such as the *owner_id* which defaults to the authorized role for the session. To remove the ViewModel, a `remove` query is issued (line 9) by providing a JSON selector which is matched against indexes to locate the ViewModel to remove. Similarly, one or more ViewModels owned by a particular *Role* can be found using a `find` query (line 10).

When the button first connects to the TrIP server, it may retrieve the persistent state of the LED by issuing a `find` query including an XPath for the `led` element (line 11-13). This returns the serialized XML for the `led` to allow for current attribute values to be retrieved. To set the state of the LED, a service may issue an update query (lines 14-17) which selects the `led` element and performs an attribute modification to set the value of the `state` attribute.

To receive real-time notification when the button is pressed, services issue a `subscribe` query selecting the `button` element and specifying the `press` event (lines 18-20). Similarly,

```
 1: <thing schema="button">
 2:   <button name="button1">
 3:     <led name="led1" state="on"></led>
 4:   </button>
 5: </thing>
 6: [ "thing.insert",
 7:   { "name":"my-button", "schema":"button",
 8:     "xml": ... } ]
 9: [ "thing.remove", { "name":"my-button" } ]
10: [ "thing.find", { "owner_id": ... } ]
11: [ "thing.find",
12:   { "name": "my-button",
13:     "\$path": "//button/led[@name=’led1’]" } ]
14: [ "thing.update",
15:   { "name": "my-button",
16:     "\$path": "//button/led[@name=’led1’]" },
17:   { "\$set": { "state": "on" }} ]
18: [ "thing.subscribe",
19:   { "name": "my-button", "\$path": "//button" },
20:   "press-event" ]
21: [ "thing.publish",
22:   { "name": "my-button", "\$path": "//button" },
23:   "press-event", ... ]
```

Fig. 5. Query examples

the button publishes a *press* event specifying the `button` element along with additional arguments, such as duration or whether it was a single or double press (lines 21-23).

V. Evaluation

A proof-of-concept implementation of TrIP has been developed to investigate transreality interactions. We present results from a benchmark evaluation gauging the responsiveness of the platform under varying concurrency and types of load. We also present a demonstration of a transreality environment implemented using the platform. Both the benchmarks and demonstration rely on queries similar to the button ViewModel query examples shown previously.

A. Proof-of-Concept Implementation

We have developed a proof-of-concept TrIP implementation using Javascript and Node.js [50]. While not optimal for a large-scale deployment, the event-driven Node.js framework is highly scalable and well suited for real-time message oriented systems such as TrIP. The present implementation utilizes an in-memory object database with log-based persistence to manage the XML DOM and metadata associated with ViewModel objects (Roles and session state are persisted using the same facilities). AVL-tree indexes over ViewModel metadata are constructed from the persistent log upon startup and used during query processing to quickly select ViewModels for operations and detect unique constraint violations. Simple two-phase locking provides concurrency control to prevent conflicts among asynchronous modification queries; locking is not performed for read-only queries like `find` and `publish` queries. In addition to XML DOM event dispatching, simple events may be published/subscribed on the ViewModel itself, this includes a reserved *mutation* event used for synchronization amongst clients.

TrIP clients have been implemented for Javascript (Node.js and web browser with WebSockets), C#, and Unity3D. Transreality *shells* which render DVMs on a particular platform have been developed for web browsers, the Microsoft Hololens, and the HTC Vive. The Hololens shell leverages the spatial mapping capabilities of the device to virtualize the surrounding environment, share the generated 3D meshes using a ViewModel representing the virtualized environment, and position objects relative to the mapped environment. The Vive shell renders the virtualized environment and objects situated therein. The browser-based shell provides a cross-platform GUI interface for browsing and interacting with objects in a TE, such as for remotely controlling devices or collaborating with other users in a TE if a HWD is unavailable.

B. Performance

Response time is the most important performance metric for interactive systems. User studies suggest that the response time for an interaction must be less than 100ms to appear instantaneous [47]. Our evaluation methodology seeks to quantify the performance limits of the proof-of-concept TrIP implementation under response time constraints, namely the maximum query rate and concurrency levels for varying types of load. TrIP is implemented using Node.js, which uses a single-process event-driven architecture to support highly concurrent loads. While this avoids thread switching costs typical of thread-per-connection architectures, there is an upper bound on the maximum throughput imposed by the event-loop and query processing costs. Above this bound, incoming queries are delayed and queued for processing resulting in reduced response times. Since query operations are executed synchronously, they may also block the event-loop and further reduce throughput by delaying the processing of asynchronous I/O operations.

1) Methodology: For CRUD queries the cost varies by query type and query cardinality (the number of affected objects) and consists of concurrency controls (locking, committing, and releasing objects), executing query operations (querying indexes, filtering objects, modifying objects, updating indexes, and persisting changes), and sending responses with query results. For publish queries the processing cost is relatively small and consists of a retrieval to find the target ViewModel and either a subscription lookup or dispatching an event through the ViewModel DOM which is handled by subscribers. However, if the number of subscribers for an event (the cardinality) is large or there are large numbers of concurrent events being published the outbound message queue can quickly become saturated and significantly reduce responsiveness.

To quantify the query processing costs and the responsiveness of the TrIP implementation, the mean throughput and response time were measured for a number of different query types and varying cardinalities and concurrencies. Six query types were evaluated with cardinalities of $n = 1$ and $n = 10$: CRUD queries (`insert`, `find`, `update`, `remove`), and `publish` queries against a ViewModel and against an XML DOM node. For each query type and cardinality, 200 simulated TrIP clients issue a collective total of 800 queries at concurrencies varying from 3 to 180. Clients are selected to issue a query in round-robin fashion (up to the concurrency limit) as soon as a client completes a query. For each variation the TrIP server is restarted and initialized with 1000 ViewModels. Each client operates on a mutually exclusive set of ViewModels created during warm-up after each client connects. For `publish` queries the cardinality sets the total number of subscribers on each ViewModel and each client publishes an event on a single ViewModel. Published events are echoed back to the publishing client after being sent to all other subscribers to measure the round trip time (RTT) for each event. Throughput and response time statistics are collected over 5 runs of each variation. The TrIP server ran in one process while clients were simulated by a Node.js load generator running in another process. The evaluation was performed on a single host with dual Intel Xeon E5335 2.00GHz CPUs (8 cores total) with 16 GB RAM running Ubuntu 14.04.5.

2) Results: The results of the evaluation are shown in Fig. 6. The decreasing publish query throughput is unexpected and is due to limitations of the load generator caused by incoming events interfering with publish query responses and causing a reduction in the throughput. Both CRUD and publish query response times increase linearly with concurrency as expected. CRUD query throughput degrades by more than half when the cardinality increases, while publish query throughput only decreases slightly. Corresponding increases in query response times are also observed. For both query types, the 100ms responsiveness constraint is exceeded at relatively low concurrency levels, restricting the present implementation to use with small-scale transreality environments where less than ≈ 10 concurrent queries are expected at any time.

It is clear that there are scalability issues with the proof-of-concept implementation that must be addressed as the platform is optimized to support larger and more complex transreality environment. However, the performance is adequate for supporting initial investigations into transreality interaction using small-scale transreality environments. In these cases the concurrency and overall query rates are expected to be relatively low as most interactions with either Embedded or Interactive Things are bursty and transient. One exception is devices providing real-time spatial tracking which will continuously update ViewModels as device or objects locations change in the environment. Supporting more than ≈ 5 of these devices in a transreality environment will require future optimization to increase publish query throughput and reduce mean event RTT. It is important to note that very little optimization was performed during development. There are many avenues available to optimize critical paths and increase performance beyond the baseline results provided by this proof-of-concept.

C. Demonstration

Using the proof-of-concept implementation of TrIP we demonstrate a simple transreality environment in which users can interact with a virtualized button in AR using a Microsoft

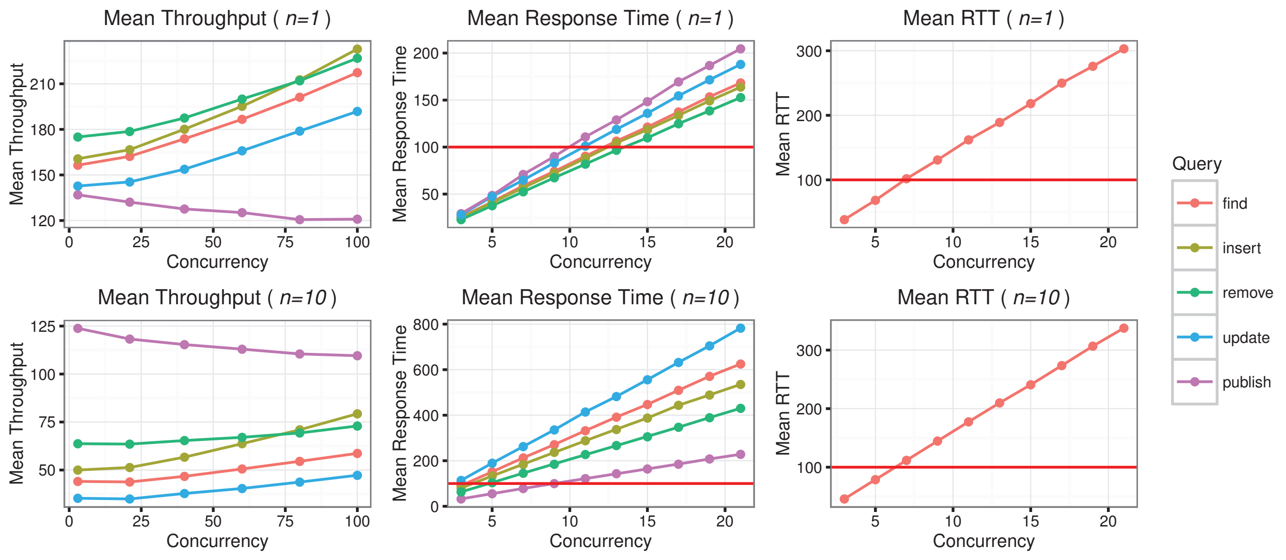

Fig. 6. Evaluation results for CRUD and publish queries with cardinalities $n = 1$ and $n = 10$. CRUD query throughput decreases with increasing cardinality while publish query throughput only decreases slightly. For $n = 1$, response times exceed the 100ms responsiveness constraint above 10 concurrent queries for both CRUD and event operations. For $n = 10$, nearly all CRUD queries exceed the constraint. Event RTTs exceed the 100ms constraint above ≈ 7 concurrent queries for $n = 1$ and above ≈ 6 for $n = 10$.

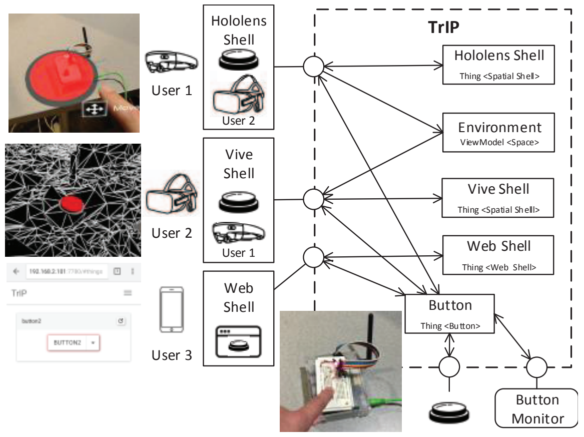

Fig. 7. System components of the proof-of-concept demonstration. TrIP clients are shown along with associated DVMs. The objects depicted in the shells represent objects visible to each user. Images show the virtualized button as perceived in each modality (counterclockwise from top left): in AR a virtual button is overlaid on the physical button, in VR the button is situated in the VE, for the web the button is represented as a GUI, finally we show the physical button used in the demo.

Hololens, in VR using an HTC Vive, remotely using a web-based GUI, and physically by directly pressing the button. Users can observe each other within the same VE as though physically present, allowing a user in VR to see an AR user standing next to them and "physically" pass a virtual object to the AR user and vice versa. The virtualized button is implemented using a Node.js client running on a Linux-based embedded computer using GPIO to control an LED and detect button presses. A button monitoring service observes the state of the virtualized button and responds to presses of the button by flashing its LED. A system diagram for the proof-of-concept demonstration in Fig. 7 depicts the DVMs used, their types and schemas, the relationships between them, and the associated frontend and backend clients. Images in the diagram depict how the button is perceived in each modality provided by the frontend clients.

The *Hololens Shell*, *Vive Shell*, and *Button* DVMs are assigned spatial locations and orientations relative to the spatial mapping of the environment generated by the Hololens and stored in the *Environment* DVM. The *Vive Shell* retrieves the spatial mapping from the *Environment* DVM to display the virtualized environment in VR. Both the Hololens and Vive shells display an avatar representing the position and orientation of other *Spatial Shell* DVMs, and a 3D representation of the virtualized button within the VE. The *Web Shell* schema does not support spatial locations, so it does not appear situated in the VE when viewed through the Hololens or Vive. However, through the web shell, a user can browse for DVMs matching the *Button* schema and display a GUI interface for interacting with the button. Note that the *Environment* and *Button* DVMs are shared and synchronized across frontend and backend clients, and that the application logic of the *Button Monitor* service is completely decoupled from the multi-modal views presented through the shells and the virtualized button. Implementing even this simple system using traditional UI design and architectural techniques would have required significant development effort. As support for new platforms is added, development costs would increase and would be largely duplicated for every subsequent application developed.

VI. Future Work

Transreality is an extremely deep concept with many facets to be explored in future work. The proof-of-concept platform provides a basis for studying transreality interactions and discovering novel interactions not possible in previous interactive environments. Many of these novel interactions rely heavily on spatial context, which currently is embedded within the ViewModel XML. Extending the object model to include spatial metadata and implementing spatial indexes over that metadata can allow for complex spatial relationships to be queried, such as proximity, nearest neighbors, and relative orientations. As we develop more complex transreality systems, ViewModel composition will become necessary to represent more complex relationships between ViewModels, such as when an object is displayed within a shell, or when a shared configuration is applied to multiple Things. While it is possible to define compositions within XML, not every schema will support the same syntax, and it will be necessary to define compositions in out-of-band metadata. The current implementation does not support efficient XPath queries over ViewModels, as it relies on a scan operation to apply the XPath on each ViewModel. Indexing XML elements will allow XPath queries to be efficiently performed over large subsets of ViewModels, enabling pattern based queries for discovering ViewModels with particular structures, such as finding all objects which provide a notification modality within proximity of a user. These are just a few possible directions for future research and development on TrIP.

VII. Conclusion

Transreality environments are highly complex interactive systems which will become more prevalent in the coming years as interactive systems and the Internet of Things converge. Service ecosystems leverage the virtualized environment provided by *Embedded Things* and the interface modalities provided by *Interactive Things* to situate virtual objects alongside virtualized physical objects, which allows for novel ad-hoc interactions between humans, virtual, and physical objects. To explore the space of transreality interactions and foster further research, we developed a proof-of-concept Transreality Interaction Platform: a specialized event-driven object database based on the Distributed ViewModel pattern which provides a novel architecture for decoupling service logic from UI implementation to simultaneously support a wide range of interface modalities. We evaluated the platform for a number of common query types to determine responsiveness under varying loads and concurrency levels, and we demonstrated the platform for integrating embedded and interactive things for seamless physical/virtual interaction in a simple trasreality environment. This work provides a foundation for future research into transreality interaction and exploration of novel and compelling interactions which span physical and virtual realities.

References

[1] S. Sorrell, "IOT - internet of transformation," Juniper Research, White Paper, Jul. 2015.

[2] A. Whitmore, A. Agarwal, and L. D. Xu, "The internet of Things—A survey of topics and trends," *Information Systems Frontiers*, vol. 17, no. 2, pp. 261–274, Mar. 2014. [Online]. Available: http://link.springer.com/article/10.1007/s10796-014-9489-2

[3] P. Montuschi, A. Sanna, and G. Paravati, "Human-Computer interaction: Present and future trends," *Computing Now*, vol. 7, no. 9, Sep. 2014. [Online]. Available: http://www.computer.org/web/computingnow/archive/september2014

[4] J. Kim, J. Lee, J. Kim, and J. Yun, "M2M service platforms: Survey, issues, and enabling technologies," *IEEE Communications Surveys Tutorials*, vol. 16, no. 1, pp. 61–76, 2014.

[5] L. D. Xu, W. He, and S. Li, "Internet of things in industries: A survey," *IEEE Transactions on Industrial Informatics*, vol. 10, no. 4, pp. 2233–2243, Nov. 2014.

[6] L. Atzori, A. Iera, and G. Morabito, "The internet of things: A survey," *Computer Networks*, vol. 54, no. 15, pp. 2787–2805, Oct. 2010. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128610001568

[7] J. Mineraud, O. Mazhelis, X. Su, and S. Tarkoma, "A gap analysis of Internet-of-Things platforms," *arXiv:1502.01181 [cs]*, Feb. 2015, arXiv: 1502.01181. [Online]. Available: http://arxiv.org/abs/1502.01181

[8] K. Ashton, "That 'internet of things' thing," *RFiD Journal*, vol. 22, no. 7, p. 97–114, 2009. [Online]. Available: http://www.itrco.jp/libraries/RFIDjournal-ThatInternetofThingsThing.pdf

[9] K. A. Hua, R. Peng, and G. L. Hamza-Lup, "WISE: a Web-Based intelligent sensor explorer framework for publishing, browsing, and analyzing sensor data over the internet," in *Web Engineering*, ser. Lecture Notes in Computer Science, N. Koch, P. Fraternali, and M. Wirsing, Eds. Springer Berlin Heidelberg, Jan. 2004, no. 3140, pp. 568–572. [Online]. Available: http://link.springer.com/chapter/10.1007/978-3-540-27834-4_69

[10] R. Peng, K. A. Hua, H. Cheng, and F. Xie, "An internet framework for pervasive sensor computing," *Int. J. Adv. Pervasive Ubiquitous Comput.*, vol. 1, no. 3, p. 1–22, Jul. 2009. [Online]. Available: http://dx.doi.org/10.4018/japuc.2009090801

[11] K. Akpınar, K. A. Hua, and K. Li, "ThingStore: a platform for internet-of-things application development and deployment," in *Proceedings of the 9th ACM International Conference on Distributed Event-Based Systems*, ser. DEBS '15. New York, NY, USA: ACM, 2015, p. 162–173. [Online]. Available: http://doi.acm.org/10.1145/2675743.2771833

[12] P. Fremantle, B. Aziz, J. Kopecky, and P. Scott, "Federated identity and access management for the internet of things," in *2014 International Workshop on Secure Internet of Things (SIoT)*, Sep. 2014, pp. 10–17.

[13] T. L. Koreshoff, T. Robertson, and T. W. Leong, "Internet of things: A review of literature and products," in *Proceedings of the 25th Australian Computer-Human Interaction Conference: Augmentation, Application, Innovation, Collaboration*, ser. OzCHI '13. New York, NY, USA: ACM, 2013, p. 335–344. [Online]. Available: http://doi.acm.org/10.1145/2541016.2541048

[14] P. Milgram and F. Kishino, "A taxonomy of mixed reality visual displays," *IEICE TRANSACTIONS on Information and Systems*, vol. 77, no. 12, pp. 1321–1329, 1994.

[15] S. Mann, "Wearable, tetherless, computer-mediated reality (with possible future applications to the disabled)," MIT Media Lab, Technical report 260, 1994.

[16] C. A. Lindley, "Trans-Reality gaming," in *Proceedings of the 2nd Annual International Workshop in Computer Game Design and Technology*, Liverpool John Moores University, UK, 2004, p. 15–16.

[17] ——, "Game space design foundations for trans-reality games," in *Proceedings of the 2005 ACM SIGCHI International Conference on Advances in Computer Entertainment Technology*, ser. ACE '05. New York, NY, USA: ACM, 2005, p. 397–404. [Online]. Available: http://doi.acm.org/10.1145/1178477.1178569

[18] M. Kranz, P. Holleis, and A. Schmidt, "Embedded interaction: Interacting with the internet of things," *IEEE Internet Computing*, vol. 14, no. 2, pp. 46–53, Mar. 2010.

[19] P. Belimpasakis and R. Walsh, "A combined mixed reality and networked home approach to improving user interaction with consumer electronics," *IEEE Transactions on Consumer Electronics*, vol. 57, no. 1, pp. 139–144, Feb. 2011.

[20] Paúl E. Estrada-Martínez, Jorge Álvarez-Lozano, J. Antonio García-Macías, and Jesús Favela, "The sentient visor: Towards a browser for the internet of things," *Intl. Symposium on Ubiquitous Computing and Ambient Intelligence (UCAmI)*, 2011.

[21] P. E. Estrada–Martinez and J. A. Garcia–Macias, "Semantic interactions in the internet of things," *International Journal of Ad Hoc and Ubiquitous Computing*, vol. 13, no. 3-4, pp. 167–175, Jan. 2013. [Online]. Available: http://www.inderscienceonline.com/doi/abs/10.1504/IJAHUC.2013.055464

[22] M. J. Kim, J. H. Lee, X. Wang, and J. T. Kim, "Health smart home services incorporating a MAR-based energy consumption awareness system," *Journal of Intelligent & Robotic Systems*, vol. 79, no. 3-4, pp. 523–535, Sep. 2014. [Online]. Available: http://link.springer.com/article/10.1007/s10846-014-0114-x

[23] T. Leppänen, A. Heikkinen, A. Karhu, E. Harjula, J. Riekki, and T. Koskela, "Augmented reality web applications with mobile agents in the internet of things," in *2014 Eighth International Conference on Next Generation Mobile Apps, Services and Technologies (NGMAST)*, Sep. 2014, pp. 54–59.

[24] B. Pokrić, S. Krc?o, and M. Pokrić, "Augmented reality based smart city services using secure IoT infrastructure," in *2014 28th International Conference on Advanced Information Networking and Applications Workshops (WAINA)*, May 2014, pp. 803–808.

[25] M. Back, D. Kimber, E. Rieffel, A. Dunnigan, B. Liew, S. Gattepally, J. Foote, J. Shingu, and J. Vaughan, "The virtual chocolate factory: Building a real world mixed-reality system for industrial collaboration and control," in *2010 IEEE International Conference on Multimedia and Expo (ICME)*, Jul. 2010, pp. 1160–1165.

[26] P. Coulton, D. Burnett, A. Gradinar, D. Gullick, and E. Murphy, "Game design in an internet of things," *Transactions of the Digital Games Research Association*, vol. 1, no. 3, Sep. 2014. [Online]. Available: http://todigra.org/index.php/todigra/article/view/19

[27] P. Coulton, "Playful and gameful design for the internet of things," in *More Playful User Interfaces*, ser. Gaming Media and Social Effects, A. Nijholt, Ed. Springer Singapore, 2015, pp. 151–173, DOI: 10.1007/978-981-287-546-4_7. [Online]. Available: http://link.springer.com/chapter/10.1007/978-981-287-546-4_7

[28] C. Lu, "IoT-enhanced and bidirectionally interactive information visualization for Context-Aware home energy savings," in *2015 IEEE International Symposium on Mixed and Augmented Reality - Media, Art, Social Science, Humanities and Design (ISMAR-MASH'D)*, Sep. 2015, pp. 15–20.

[29] A. MacDowell and M. Endler, "Internet of things based multiplayer pervasive games: An architectural analysis," in *Internet of Things. User-Centric IoT*, ser. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, R. Giaffreda, R. Vieriu, E. Pasher, G. Bendersky, A. J. Jara, J. J. P. C. Rodrigues, E. Dekel, and B. Mandler, Eds. Springer International Publishing, Oct. 2014, no. 150, pp. 125–138, DOI: 10.1007/978-3-319-19656-5_19. [Online]. Available: http://link.springer.com/chapter/10.1007/978-3-319-19656-5_19

[30] V. C. Stoianovici, D. Talaba, A. V. Nedelcu, M. M. Pisu, F. Barbuceanu, and A. Stavar, "A virtual reality based human-network interaction system for 3D internet applications," in *2010 12th International Conference on Optimization of Electrical and Electronic Equipment (OPTIM)*, May 2010, pp. 1076–1083.

[31] J. T. Carvalho, R. A. P. d. Santos, S. S. d. C. Botelho, N. D. Filho, R. R. Oliveira, and E. Santos, "Hyper-Environments: a different way to think about IoT," in *Internet of Things (iThings/CPSCom), 2011 International Conference on and 4th International Conference on Cyber, Physical and Social Computing*, Oct. 2011, pp. 25–32.

[32] A. Ricci, L. Tummolini, M. Piunti, O. Boissier, and C. Castelfranchi, "Mirror worlds as agent societies situated in mixed reality environments," in *Coordination, Organizations, Institutions, and Norms in Agent Systems X*, ser. Lecture Notes in Computer Science, A. Ghose, N. Oren, P. Telang, and J. Thangarajah, Eds. Springer International Publishing, May 2014, no. 9372, pp. 197–212, DOI: 10.1007/978-3-319-25420-3_13. [Online]. Available: http://link.springer.com/chapter/10.1007/978-3-319-25420-3_13

[33] A. Ricci, A. Croatti, P. Brunetti, and M. Viroli, "Programming mirror worlds: An Agent-Oriented programming perspective," in *Engineering Multi-Agent Systems*, ser. Lecture Notes in Computer Science, M. Baldoni, L. Baresi, and M. Dastani, Eds. Springer International Publishing, May 2015, no. 9318, pp. 191–211, DOI: 10.1007/978-3-319-26184-3_11. [Online]. Available: http://link.springer.com/chapter/10.1007/978-3-319-26184-3_11

[34] C. W. Thompson, "Next-Generation virtual worlds: Architecture, status, and directions," *IEEE Internet Computing*, vol. 15, no. 1, pp. 60–65, Jan. 2011.

[35] M. Macik, T. Cerny, and P. Slavik, "Context-sensitive, cross-platform user interface generation," *Journal on Multimodal User Interfaces*, pp. 1–13, Feb. 2014. [Online]. Available: http://link.springer.com/article/10.1007/s12193-013-0141-0

[36] I. Dalmasso, S. Datta, C. Bonnet, and N. Nikaein, "Survey, comparison and evaluation of cross platform mobile application development tools," in *Wireless Communications and Mobile Computing Conference (IWCMC), 2013 9th International*, Jul. 2013, pp. 323–328.

[37] A. Syromiatnikov and D. Weyns, "A journey through the land of Model-View-Design patterns," in *2014 IEEE/IFIP Conference on Software Architecture (WICSA)*, Apr. 2014, pp. 21–30.

[38] N. Elmqvist, "Distributed user interfaces: State of the art," in *Distributed User Interfaces*, ser. Human-Computer Interaction Series, J. A. Gallud, R. Tesoriero, and V. M. R. Penichet, Eds. Springer London, Jan. 2011, pp. 1–12. [Online]. Available: http://link.springer.com/chapter/10.1007/978-1-4471-2271-5_1

[39] H. Jetter, M. Zöllner, J. Gerken, and H. Reiterer, "Design and implementation of Post-WIMP distributed user interfaces with ZOIL," *International Journal of Human-Computer Interaction*, vol. 28, no. 11, pp. 737–747, Nov. 2012. [Online]. Available: http://dx.doi.org/10.1080/10447318.2012.715539

[40] T. Seifried, H. Jetter, M. Haller, and H. Reiterer, "Lessons learned from the design and implementation of distributed Post-WIMP user interfaces," in *Distributed User Interfaces*, ser. Human-Computer Interaction Series, J. A. Gallud, R. Tesoriero, and V. M. R. Penichet, Eds. Springer London, 2011, pp. 95–102, DOI: 10.1007/978-1-4471-2271-5_11. [Online]. Available: http://link.springer.com/chapter/10.1007/978-1-4471-2271-5_11

[41] M. A. Fernández, V. Peláez, G. López, J. L. Carus, and V. Lobato, "Multimodal interfaces for the smart home: Findings in the process from architectural design to user evaluation," in *Ubiquitous Computing and Ambient Intelligence*, ser. Lecture Notes in Computer Science, J. Bravo, D. López-de-Ipiña, and F. Moya, Eds. Springer Berlin Heidelberg, Dec. 2012, no. 7656, pp. 173–180, DOI: 10.1007/978-3-642-35377-2_24. [Online]. Available: http://link.springer.com/chapter/10.1007/978-3-642-35377-2_24

[42] L. Riliskis, J. Hong, and P. Levis, "Ravel: Programming iot applications as distributed models, views, and controllers," in *Proceedings of the 2015 International Workshop on Internet of Things towards Applications*. ACM, 2015, p. 1–6. [Online]. Available: http://dl.acm.org/citation.cfm?id=2820977

[43] J. Guerrero-Garcia, J. Gonzalez-Calleros, J. Vanderdonckt, and J. Muoz-Arteaga, "A theoretical survey of user interface description languages: Preliminary results," in *Web Congress, 2009. LA-WEB '09. Latin American*, Nov. 2009, pp. 36–43.

[44] C. Perera, A. Zaslavsky, P. Christen, and D. Georgakopoulos, "Context aware computing for the internet of things: A survey," *IEEE Communications Surveys Tutorials*, vol. 16, no. 1, pp. 414–454, 2014.

[45] M. Maheswaran, J. Wen, and A. Gowing, "Design of a context aware object model for smart spaces, things, and people," in *2015 IEEE International Conference on Communications (ICC)*, Jun. 2015, pp. 710–715.

[46] M. Fowler, *Patterns of Enterprise Application Architecture*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2002.

[47] J. Nielsen, *Usability Engineering*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993.

[48] "Document object model (DOM) level 2 events specification," https://www.w3.org/TR/2000/REC-DOM-Level-2-Events-20001113/. [Online]. Available: https://www.w3.org/TR/2000/REC-DOM-Level-2-Events-20001113/

[49] "W3C DOM4," https://www.w3.org/TR/2015/REC-dom-20151119/. [Online]. Available: https://www.w3.org/TR/2015/REC-dom-20151119/

[50] S. Tilkov and S. Vinoski, "Node.js: Using JavaScript to build High-Performance network programs," *Internet Computing, IEEE*, vol. 14, no. 6, pp. 80–83, 2010.