NooSphere: An Activity-Centric Infrastructure for Distributed Interaction

Steven Houben, Søren Nielsen, Morten Esbensen, Jakob E. Bardram
The Pervasive Interaction Technology Laboratory
IT University of Copenhagen, Rued Langgaardsvej 7, DK-2300 Copenhagen,Denmark
{shou,snielsen,mortenq,bardram}@itu.dk

ABSTRACT

Distributed interaction is a computing paradigm in which the interaction with a computer system is distributed over multiple devices, users and locations. Designing and developing distributed interaction systems is intrinsically difficult as it requires the engineering of a stable infrastructure to support the actual system and user interface. As an approach to this re-engineering problem, we introduce NooSphere, an activity-centric infrastructure and programming framework that provides a set of fundamental distributed services that enables quick development and deployment of distributed interactive systems. In this paper, we describe the requirements, design and implementation of NooSphere and validate the infrastructure by implementing three canonical real deployable applications constructed on top of the NooSphere infrastructure.

Keywords

Distributed Interaction, Activity-Centric Computing, Distributed Computing, Activity Cloud, NooSphere

Categories and Subject Descriptors

H.5.m. [**Information Interfaces and Presentation**]: Miscellaneous

General Terms

Design, Human Factors

1. INTRODUCTION

With the widespread introduction of mobile devices (such as tablets and smartphones) and increased availability of large displays (such as situated displays and tabletops), setups in which users are engaged with multiple devices at the same time are becoming more common outside of the traditional smart space environment (such as Gaia [35] or iLand [37]). Heterogeneous multi-device environments have the potential to introduce new cross-device interaction techniques, support seamless shared information spaces based on the users' tasks or provide a new platform to explore collaborative setups. More general, *distributed interaction* refers to a paradigm in which interaction with a computer system is dynamically distributed over one or multiple (i) users, (ii) devices and (iii) locations. These three elements change over time and are in an ad-hoc and dynamic way related to the tasks or activities people are doing. Information is no longer tied to one specific personal device but multiple personal devices are rather mediators to the ubiquitous personal information space provided by the internet.

MUM '13, December 02 - 05 2013, Luleå, Sweden
Copyright 2013 ACM 978-1-4503-2648-3/13/12...$15.00.
http://dx.doi.org/10.1145/2541831.2541856

However, as pointed out by Edwards et al. in "*The infrastructure problem in HCI*", applications and user interfaces are not designed in isolation but on top of toolkits and infrastructures [15]. In fact, underlying technology determines to a great extend the capabilities and limitations of the interactive application. The problems with designing systems for distributed interaction lies in the fact that support for a large number of important distributed interconnected services needs to be in place. In an environment of changing users, devices and locations, these requirements pose a challenging and time consuming task to developers. First, there are a large number of problems related to the storage, sharing, replication, synchronization and contextualization of *data and information*. In order to provide seamless support for these, developers need to deal with complex dynamic network setups, synchronization between network protocols and concurrency issues. Second, there are a number of challenges related to the discovery and pairing of heterogeneous devices as well as the integration of distributed interaction systems in application-oriented platforms. Designing, prototyping and developing stable distributed interaction applications that are deployable in the wild is extremely challenging because it requires developers to engineer a stable infrastructure to support a number of distributed services that support the user interfaces. Moreover, the complexity of defining and managing a data and context model that provides the appropriate support for distributed interaction systems, is intrinsically tied to the architecture of the underlying infrastructure.

As an approach to this re-engineering problem and exploration into infrastructure design, we introduce *NooSphere*, an activity-centric infrastructure that provides a flexible platform for the prototyping of distributed interaction systems. We report of the architecture and components of NooSphere and describe three canonical applications built on top of NooSphere. The paper concludes with a reflection on the merits and limitations of our approach.

2. RELATED WORK

Smart space approaches have originally initiated the research into infrastructures and architectures that support distributed user interfaces (DUI). The seminal work of Marc Weiser [41] at Xerox PARC describing a ubiquitous vision that includes pads, tabs and large displays originated a vast body of research in interactive smart spaces. One of the earliest smart spaces is iLand [37] which used the roomware BEACH to create shared information spaces that stretch different displays and devices. Another classic smart space system, iRoom [22], provides the ability for pointer redirection, content replication and collaboration through an infrastructure- centric approach based on an event and data heap.

Project Aura [16] is a pervasive smart space system based on an adaptive infrastructure that supports surrogate clients that amplify the capabilities of mobile devices, nomadic file systems through data staging and network advisors. Aris [8] supports the redirection of windows and input between different types of devices including private devices and public displays in an effort to support a multi-user legacy user interface environment. Other approaches that provide support for pointer redirection are Pointright [23] and Impromptu [9]. *XICE* allows for the extension of input and output of mobile devices by *annexing* it to a smart space wall or table displays [1]. The Gaia Operating system [35] is a meta operating system that provides support for the coordination of software entities distributed over heterogeneous networked devices contained inside a physical space. Finally, *Shared Substances* [17] is a novel data oriented middleware that proposes to decouple functionality from data to support the design of multi-surface applications. A number of systems have been proposed to move beyond the smart room into smart buildings. ReticularSpaces [5], e.g., is built on top of a peer-to-peer event system that supports distributed hash maps to share data between different devices. More recently, the Window Brokers system [2] introduced an approach to annex devices into a shared workspace using display servers.

In literature, many existing approaches to pervasive middleware have been described over the years (including [6, 10, 12]). Several approaches focus specifically on collaboration inside a pervasive environment using platform compositions [33], context-awareness and semantic technologies [39], missions [24], proxy devices [38], services [34] and roles [18]. Ecora [32] is an agent-based pervasive framework for the construction of context-aware applications that focuses on heterogeneity, scalability, communication and usability. A similar infrastructure is GlobeCon [27] which provides support for distributed and pervasive computing in large-scale environments, thus moving beyond rooms or buildings. For a complete overview and classification of context-aware systems, we refer to [19] and [3]. Recently, a number of cloud platforms [25] including Google Apps Engine, Microsoft Azure and Amazon S3, have empowered developers to create applications in which data and services transcend the individual device and can be consumed on any device with internet access. This model adheres much more to the idea of a personal information cloud [28].

Most of these smart space systems and pervasive infrastructures however, suffer from three main issues. First, they are contained in one physical environment (room or building), neglecting some of the impact of mobility and interconnectivity. Second, most of these systems are extremely complex to deploy and do not integrate with existing applications and platforms, putting a great strain on developers and users. Finally, because these systems transcend individual devices, they introduce context or aggregation models to support end-users. However, many of these models are arbitrary context models and do not reflect the tasks or activities people do with these systems. NooSphere draws from this previous work to provide a lightweight infrastructure and programming framework that unifies the *interconnectivity* of cloud platforms with the *dynamics* of smart room technology. The core contribution of this paper is a novel generic and reusable infrastructure that represents *data and context* in an *activity-centric approach*, which reflects the actual tasks people do with these systems. Informed by prior studies on activity-centric computing approaches [5, 20], the infrastructure and all its services are thus designed specifically around this notion of *activity*.

3. REQUIREMENTS

Based on the related work discussed above and prior research into approaches for distributed environments, we have derived 7 core requirements for distributed interaction systems:

R1: **Persistence** – the infrastructure should provide a persistence mechanism that can be used to store data and information from any location, device type, or context. This will allow data and information to transcend the local space and context and become a truly ubiquitous concept based on an extensive life cycle re-use as described by Moran and Zhai [28]. Additionally, to support persistence during offline sessions, the infrastructure should cache data and events locally.

R2: **Distribution** – the infrastructure should support *synchronous* and *asynchronous* distribution of data models and files both in a local space as well as outside of this space. This allows for the connection of different distributed systems from different domains into one shared distributed interaction space.

R3: **Discovery and Pairing** – to allow applications and devices to seamlessly join a distributed space, the system must provide built-in support for (i) discovery of services, and (ii) an automatic pairing system that annexes different devices or applications into one seamless space. Since computer use is shifting from device specific applications and files to cross-device information and services, support for aggregation and composition of multiple devices should be an inherent part of the infrastructure.

R4: **Coordination and Communication** – as information and data is increasingly being used at the same time by multiple users, the infrastructure should support the notion of *user multiplicity* by allowing the attachment of specific artifacts, events, workflows or messages to the underlying data model. This ensures that applications or devices can support multi-user coordination and communication tools.

R5: **Configuration** – configuration refers to the process of determining the state of an application or device. To seamlessly move information between different applications or devices, the infrastructure should support the configuration of information on one or multiple devices for one or multiple users. This will allow users to manage data and services on different devices or applications.

R6: **Context Handling** – to support the use of context-aware functionality (e.g. location trackers) or embedded systems (such as Gadgeteer or Arduino), the infrastructure should provide a mechanism for system specific context processors that allow for the distribution of context information over all connected applications and devices.

R7: **Interoperability** – to fully support a multi-device configuration, the infrastructure should provide support for different operating systems and platforms by implementing a platform independent protocol. This will allow future inclusion of new technologies or platforms.

In order to support quick prototyping, development and deployment of distributed interaction systems, the requirements should be implemented with an appropriate level of abstraction, leaving e.g. networking and file management transparent to the developer. At the same time, the infrastructure should be extensible to meet new requirements. NooSphere is a lightweight and scalable toolkit that implements these requirements and can be used to design, prototype and develop interconnected activity systems for distributed interaction.

4. NOOSPHERE

NooSphere is an activity-centric service-based infrastructure and programming framework to support the development and deployment of distributed interactive systems. It is built on the concept of *communicating activity systems* [21]: by using a standardized data model and a two layered infrastructure consisting of a cloud and local distributed system, it allows for the deployment of different distributed interaction applications that can be interconnected. This implies that data and services are not confined within one system but can be consumed in all interconnected systems through adaptation of the context. This allows developers to built very complex distributed applications that consist of different domain specific system that are interconnect through the cloud. As illustrated in Figure 1, NooSphere is therefore composed of two fundamental components: (i) NooCloud, a cloud platform, and (ii) NooSystem, a distributed activity system.

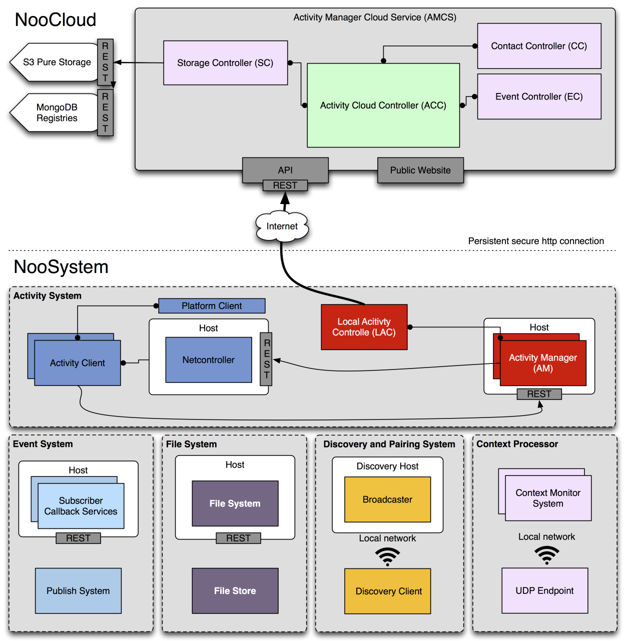


Figure 1: The architecture of NooSphere is composed of two components: (i) NooCloud, a cloud infrastructure that supports data storage, event distribution and activity management, and (ii) NooSystem, a dynamic distributed system that supports cross-device activity management, file and event distribution as well as a discovery and pairing mechanism.

4.1 Activity-Centric Computing

Activity-centric computing (a concept that was originally introduced by Apple Research [30]) is an interaction paradigm that provides support for the users' activities, rather than the tools they use to perform those activities. An activity is a higher level structure that encapsulates all resources and tools relevant to a specific task in order to represent an intention of work. Over the years several approaches to activity-centric computing have been successfully deployed to support (i) desktop multitasking, (ii) context handling and (iii) augmented interaction. The paradigm has been applied to different areas in Human-Computer Interaction (HCI), ranging from task management to collaborative work on the desktop interface [4, 13, 20, 29, 40]. However, more recently the approach has also been applied to distributed user interfaces and pervasive computing [5, 7, 11, 26] demonstrating its merit as a context model in a multi-device environment. Based on the experiences and lessons learned from successfully deploying these activity-centric systems, we propose to move to a more generalized activity-centric infrastructure, which will allow for more advanced prototyping and deployment.

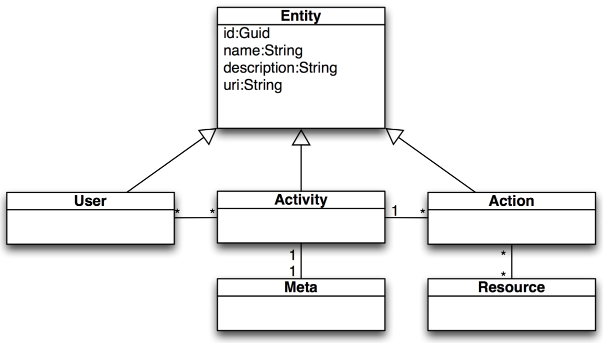


Figure 2: NooSphere uses activity as a first class object in all operations. The minimal activity object is composed of (i) users, (ii) meta information and (iii) actions, which are subtasks that contain resources such as files or links to web services..

4.2 Activity Model

The main advantage of activity-centric computing is that the activity model is a reflection of real physical tasks and activities that people do. Compared to other arbitrary context models (like those discussed in the related work section), the close mapping between the intention of the user's interaction and the digital representation of that intention allows users to easily use, appropriate and configure the model thereby minimizing intelligibility problems [5, 20]. The activity model includes *information* (such as files and resources), *coordination information* (such as users and roles) and *configuration states* (such as application descriptions) making it a well suited match for a multi-user, multi-device and multi-location environment. The first class object in the NooSphere infrastructure is thus an *activity*, as illustrated in Figure 2. In its minimal configuration, an activity is composed of (i) users, (ii) actions which contains resources and (iii) meta data.

Users. Users are digital representations of human agents that interact with the activity. They are part of the activity, they own, shape, define, consume, and share activities by interacting with the system. While the activity has one owner (the creator of the activity), it can be accessed by multiple users based on roles or other limitations imposed on the user object. All changes to the activity are shared with all users associated with the activity. The *user multiplicity* is an inherent part of the infrastructure and can thus be used to determine how actions and resources of an activity should be shared or consumed by the application.

Actions. Each activity is subdivided into a set of actions, which are tasks that are part of the activity. Actions structure how users interact with the different resources, such as files, folders and web services. Additionally, actions can be modelled as workflows, which are structured or unstructured sequences that are imposed or defined by the user. Actions thus describe *functions of work* as well as resources that are part of the activity.

Meta Data. Each event that occurs within the activity is logged and stored into the activity itself for persistence and reflection. The history can be used to track changes in parts the activity, create awareness on different actions or simply to visualize the development. Each activity is uniquely defined by an identity which consists of meta data such as a name, image or description and a unique reference number (e.g. GUID). An activity can be connected to other activities creating hierarchical relationships or references between activities.

4.3 NooCloud

NooCloud is an event-driven cloud-based service platform that supports (i) persistence of the activity model, (ii) storage capabilities for files, events and activity models, and (iii) cloud-based event distribution. The main component of NooCloud is the activity cloud controller (ACC), a cloud infrastructure exposed by a RESTful HTTP API. The ACC is the entry point of the framework and provides a number of activity-centric services grouped into a set of controllers, based on the Front Controller design pattern. Each controller is responsible for every create, read, update and delete (CRUD) operation performed upon its certain area of responsibility and related tasks are thus forwarded by the ACC to the correct controller. All controllers are aggregated in the ACC and exposed as an activity manager cloud service (AMCS). This service accepts HTTP requests for all functionality, ranging from file to activity management and is thus the public access point. Through the API, users can create, read, update and delete activity models and containing resources through pure HTTP requests. On top of this service runs a public website, which has two main purposes: (i) providing documentation on the API and (ii) allowing users to create an account or manage existing accounts.

4.3.1 Storage Controller

The storage controller (SC) provides support for the storage of all the infrastructure primitives, which are entire activity models, actions, users, devices and events. The SC supports two types of storage: (i) registries, which are entries that are stored in a NoSQL database and (ii) pure storage, which is binary serialized data. Registries store basic information for quick lookup and support advanced search and retrieval. The registry supports retrieval of entire activity models, actions, devices and users based on the GUID described in the entity base class. Because the activity model in NooCloud needs to be very flexible, a pure storage is implemented that stores the serialized extensions of the data model objects as well as actual files in the object cloud storage. This means that the activity itself and all its resources are saved directly in the cloud storage while keeping a reference in the lookup registry. *(Implementing R1: persistence)*

4.3.2 Event Controller

A local activity system can connect to the NooCloud through an HTTP endpoint that is exposed by the AMCS and handled by the event controller (EC). When a user or local system connects to the AMCS, it passes the login attempt to the EC, which establishes a persistent HTTP connection and returns a connection ID if the account is found. This ID is then used by the local system as a basic authentication token and is passed on as a parameter in each of the following HTTP requests. Users can connect several devices or local systems to the same cloud account causing the EC to register the connection to multiple devices or systems. Whenever a controller (e.g. storage controller) has handled a request that either changes data, or is relevant for other users or devices, the EC is asked to notify the relevant connected devices. The event is pushed to local users or systems over the persistent HTTP connection. The EC uses a centralized key-value store to store events before forwarding them. Through the EC, the infrastructure supports synchronous updates and work on multiple devices involving multiple users. *(Implementing R2: distribution)*

4.3.3 Contact Controller

Through the AMCS, a contact controller (CC) takes care of the handling of user management. A user can prompt another user to become a contact, and a contact request is thereafter stored in the registry as well as send to the prompted user. They can then choose to accept the request, after which the two users are connected and automatically subscribed to each others activities upon connect. A contact can be removed at any time, disabling the notifications from the other user. *(Implementing R4: communication and coordination)*

4.4 NooSystem

NooSystem is a dynamic service-based infrastructure that supports the distribution of activity model instantiations, files, communication and coordination messages and context representations in a local multi-device information space. The infrastructure uses a flexible service model that during compile time can be accessed through a DLL or class files, while each service at runtime is accessible through a REST HTTP service that is hosted in its own url-based service host. The infrastructure is composed of five distributed subsystems: (i) the activity system, (ii) publish-subscribe event system, (iii) a file system, (iv) a discovery and pairing system and (v) a context monitoring system.

4.4.1 Activity System

The activity system is subdivided into two components: activity manager and activity client. The activity manager (AM) is used as a coordinator between different local clients and/or managers that are connected inside the local NooSystem and is directly connected to one account of the activity cloud controller (ACC). The activity client on the other hand, is used to consume the activities on local devices and is directly connected to a local AM, which distributes the activities to the clients.

The activity manager (AM) is typically connected to an instance of NooCloud and thus is synchronized with all activities stored in the cloud for a particular user account. The AM is connected to the ACC through a Local Activity Controller (LAC), which creates and maintains a persistent HTTP connected with NooCloud. All events that are distributed by the ACC are handled by the LAC and passed on to the AM, which distributes the events through a local distributed publish / subscribe system to all connected clients. The AM caches all activities locally to deal with network interruptions and to speed up the distribution of activities and resources. The AM is exposed to the local NooSystem through RESTful http services. The AM can also be used only in a local setting (without the cloud link) and can even be connected to the activity store of another AM. The activity client (AC) is composed of a netcontroller and a platform client. Each AC is connected to a local activity manager (AM) through the netcontroller which registers itself with that AM with a callback service address that runs a client REST HTTP service to which the AM can send distributed events. The platform client (PC) converts the netevents into native events (e.g. `C#` delegates) which are then exposed to the platform integration layer. Both the activity client (AC) and activity manager (AM) are composed of four local distributed subsystems: (i) event system, (ii) file system, (iii) discovery system and (iv) context monitor system. *(Implementing R5: configuration)*

4.4.2 Event System

The distributed event system (ES) is an HTTP REST publish/subscribe service that supports the distribution of messages, activity events, device events, user events, file events and external events. When an activity client is connected to an activity manager, the client sends an HTTP request to the manager that contains a device object, describing the device, and a callback service address on which the activity client device is running the callback services. The activity manager registers the activity client and active device and publishes all events and messages to the host address of the activity client. These events include system messages which contain either user content (such as chat messages) or control messages (such as reconnect requests). The event system also distributes changes to the activity collection (such as added, removed, changed or locked) to inform all connected clients to update the local visualization. Next, every time a device containing an activity client connects to or disconnects from the activity manager, device messages containing information about changes in the device collection are sent to all connected clients. *(Implementing R2: distribution)*

4.4.3 File System

Both the activity client and activity manager are equipped with a file system. The file system is composed of a file server, which is responsible for saving (to disk) and loading files (to stream), and a file store, which is a key-value store that registers all files that are part of the loaded activities (NoSQL database). The file service distributes file events through the event service (request to download, request to upload and deleted), when files are changed at the activity manager. The source can be both local (in the NooSystem) but also external (by an external NooSystem that is connected through the NooCloud). When a new file is added to an activity by an activity client, the activity model is first updated to the activity manager, which will in turn send a *request to upload* message to the client. When the client uploads the file to the file store of the manager, the activity manager sends an *request to download* to all other connected activity clients as well as the NooCloud. The NooCloud in its turn will request the local activity manager to upload the file to the cloud storage. Because all updates are sent to all attached activity clients and managers, local applications using the infrastructure can decide how to handle file consistency and potential conflicts. *(Implementing R1: persistence and R2: distribution)*

4.4.4 Discovery and Pairing System

Each activity manager is equipped with a broadcast service, which broadcasts the manager's name, host address and device information (type, physical location and ID) over the local network. When the broadcast service is started, a separate host with a dedicated address is launched. The broadcast service can be dynamically (re-)configured and (de-)activated at runtime, to allow developers to toggle discovery support. In order to find activity managers on a local network, both the activity client as well as activity manager are equipped with a discovery service. This service searches the local network for available activity managers and exposes them to the main controller for consumption. The current version of NooSystem implements both Webservice Discovery as well as Apple Bonjour protocols to support different types of devices and operating systems. *(Implementing R3: discovery and pairing)*

4.4.5 Context Monitor System

To add support for context-aware and embedded devices (such as Arduino or location trackers), the activity manager and activity client are equipped with a context processor. This processor tracks a collection of *IContextService* objects which are monitored in separate threads. Each event triggered by the context processor is distributed through either the event system (as a context message) or via a UDP multicast system (for real-time services) that is dynamically launched and attached to the context processor. *(Implementing R6: context handling)*

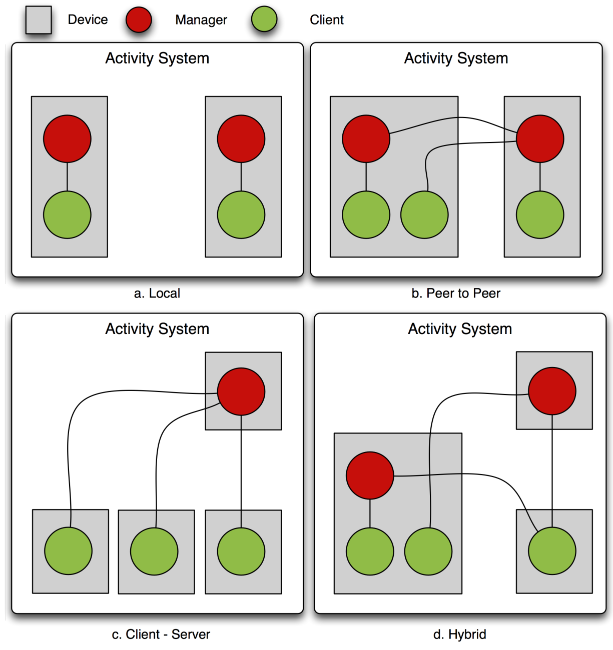


Figure 3: The local infrastructure NooSystem, can be deployed in different configurations: (a) manager and client on the same devices, (b) peer to peer connection between manager on different devices, (c) traditional client server approach with a manager on a dedicated devices or (d) hybrid setup composed of both dedicated and local managers.

4.5 Deployment

The infrastructure is designed to be modular and scalable and can thus be used in combination (NooCloud and NooSystem) but also separate. Since NooCloud provides an open API, any developer can design a custom activity system using any language or platform that supports REST HTTP calls. NooSystem on the other hand has the ability to work with local activities only, thereby removing the necessity of connecting to the cloud part to use activities. *(Implementing R7: interoperability)*

The NooSystem is also modular on a local level. Since both the client and manager are lightweight services, they can be spawned on either the same or different devices (Figure 3 a) using HTTP services. Activity managers can also connect to each other, emulating a peer to peer system (Figure 3 b). With this approach, different managers can exchange activities of different users. In case one device is dedicated for the activity manager, the setup can also be configured as client-server (Figure 3 c). Because of the service-based approach, hybrid approaches that merge dedicated managers and local managers can be connected to form one NooSystem (Figure 3 d). Finally, the code library provides the ability to use the activity manager and client directly in code without the need for dynamic REST HTTP services. Because of the architecture of the system, the network code is completely transparent for developers, who simply need to connect to a running service from within their application. All network code is made transparent by wrappers that translate network events send by the NooSystem to local delegates.

The NooCloud infrastructure is built on top of the ASP.NET Web API Framework and runs on the AppHarbour cloud platform. The event system is implemented using SignalAR in order to support a cloud-based real-time publish-subscribe mechanism. The NoSQL database used by NooCloud to create the registries is MONGODB The cloud infrastructure is exposed to the web through a REST HTTP API. The Noo System is implemented using Mono WCF (Windows Communication Foundation), Web API and runs on Windows, Linux OS X and Android. Each service (activity, discovery and file) runs in a custom built service host (using Owin) which exposes the service through a REST HTTP service. The local storage is implemented using RavenDB.

5. CASE STUDIES

To validate the *functionality* of the infrastructure and test the *stability*, *performance* and *feasibility* of the architecture, we present three canonical distributed activity-centric case studies [14] that represent real deployable and testable system similar to those found and tested in research and industry. Table 1 provides an overview of the three applications and their use of the underlying features provided by NooSphere.

Building different reference applications on top of an infrastructure has been proposed as a robust research method for evaluating infrastructures [14, 15]. These applications demonstrate the functionality of the infrastructure and can be used as input for its applicability for supporting application development. In this section, we present three such case studies, and discuss how they benefit from using the NooSphere platform. In the case studies we present:

- **Case Study #1: co-Activity Manager** – reimplementation of an existing application [20].
- **Case Study #2: Dynamic Device Composition** – construction of an application using the basic infrastructure.
- **Case Study #3: SmartWard** – rapid prototyping of an advanced research application by leveraging the features of the infrastructure.

5.1 CS #1: co-Activity Manager

Description. Task- and activity-based desktop systems have been proposed as a mean to contextualize desktop usage as they re-organize information based on tasks people do. With the more widespread availability and usage of tablets and e-readers, significant research have been investigating how to seamlessly integrate the exchange of contextual files between different types of devices and users. In this application, we reimplemented co-Activity Manager [20] (cAM), an activity-centric desktop manager using the NooSphere infrastructure and extended its functionality so it seamlessly supports resource sharing with a tablet containing an e-reading application.

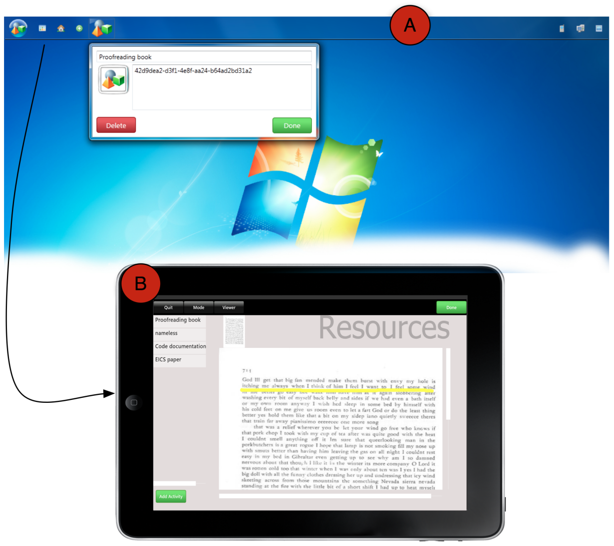
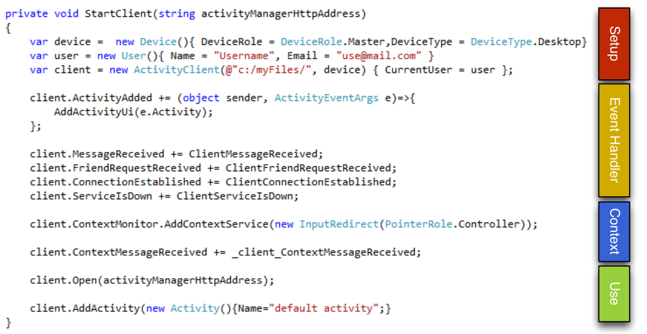


Figure 4: co-Activity Manager [20] (cAM) is a multi-user activity centric desktop interface that contains an activity bar at the top of the screen (A). Each activity button represents an ongoing activity; clicking the button will load the virtual desktop related to the activity as well as repopulate the desktop background. Files can be related to the activity by dragging and dropping them on the desktop background of a loaded activity. The tablet (B) can be paired to cAM by selecting the device from the auto-detect menu. Users can simply drag and drop files they want to use on a tablet on the activity bar located at the top of the screen.

Using the Infrastructure. Using NooSphere as underlying infrastructure, the entire network, activity management and communication code base from the original project (roughly 3000 lines of code) could be replaced by 50 lines of code required to initialize the infrastructure. The user interface and features are identical as in the original implementation. However, the reimplementation that uses the *activity client* and *activity manager* from NooSphere, simply needed to setup the activity system and hook the user interface components to the events produced by the infrastructure. Figure 5 shows the code used to set up the activity client using the NooSphere infrastructure. The code sample first demonstrates how to create and initialize a new user and device and associate them with the new activity client and activity manager. It also shows a number of example event available through the activity client object. The callbacks hooked to these events are part of the infrastructure and designed to de-serialize the JSON objects used by the infrastructure into native typed c# objects. After opening the activity client, an activity repository is created (or loaded in case a previous session is detected), a file server is started and the REST and discovery services are run automatically in the background. Because the infrastructure automatically includes support for sharing activities and their containing resources and contacts with other devices, we extended the system with a tablet application that can be used for active reading and resource browsing. The tablet application also implements an activity client, which connects automatically to the activity manager of cAM using the built-in discovery mechanism. All changes to resources are automatically synced by the

Feature	#1: Reimplementation	#2: Device Composition	#3: Extending the infrastructure
Persistence	Activities and files across devices in the cloud	Activities and files in the cloud	Local data store and file system
Distribution	Synchronisation of activities and resources	Sharing and synchronisation of files	Synchronisation of activity states
Discovery	Automatic detection of tablet through background discovery service	Ad hoc pairing triggered by vision systems	Automatic ad hoc pairing based on location /proximity
Coordination	Friend list, sharing activity models through cloud	Device control (master-slave)	Activity-centric messaging (nurses records)
Configuration	Structure desktop using activities that can be deployed on any device.	File management across multiple users and devices	Multi-user synchronized activity manager
Context	No	Pointer and touch redirect	Location tracker, Arduino support for RFID scanner
Interoperability	Abstracts activities into a shareable form (JSON)	Resources to encapsulate native files	Abstracts activities into a shareable form (JSON)

Table 1: An overview of the three canonical applications and how they each utilize different features provided by the NooSphere infrastructure.



```
private void StartClient(string activityManagerHttpAddress)
{
    var device =  new Device(){ DeviceRole = DeviceRole.Master,DeviceType = DeviceType.Desktop}
    var user = new User(){ Name = "Username", Email = "use@mail.com" }
    var client = new ActivityClient(@"c:/myFiles/", device) { CurrentUser = user };

    client.ActivityAdded += (object sender, ActivityEventArgs e)=>{
        AddActivityUi(e.Activity);
    };

    client.MessageReceived += ClientMessageReceived;
    client.FriendRequestReceived += ClientFriendRequestReceived;
    client.ConnectionEstablished += ClientConnectionEstablished;
    client.ServiceIsDown += ClientServiceIsDown;

    client.ContextMonitor.AddContextService(new InputRedirect(PointerRole.Controller));

    client.ContextMessageReceived += _client_ContextMessageReceived;

    client.Open(activityManagerHttpAddress);

    client.AddActivity(new Activity(){Name="default activity";}
}
```

Figure 5: The basic infrastructure programming framework provides developers with a number of objects that hide all the underlying complexity. This short code sample e.g. exposes all underlying network, synchronization, serialisation and context handling through basic events that send back a native usable object.

infrastructure and presented to the UI by native events. Some minor updates to the UI of cAM were required to deal with immediate updates from the tablet device (e.g. dispatch the received object to the UI thread), but no extra infrastructure code was required to add an additional device. Table 1 shows how the reimplementation of cAM uses almost all basic services except context handling, which was not a requirement for this particular project.

User Experience. The basic services provided by NooSphere allows for a seamless multi-device experience, in which users need to perform very little manual setup, or *configuration work*, in order to pair devices or share information with other devices. Because of the built-in discovery system, the user can simply pair devices by clicking a button. There is thus no need to setup shared folders, add credentials or install third party applications. Through the user interface of cAM, the user can simply drag and drop resources (such as files or contacts) on top of an activity button, which causes the infrastructure to automatically transfer it to the attached tablet that visualizes the newly added resources in context of the existing activities. The consistent use of activities as structuring mechanism thus provides users with a consistent mental model across devices.

5.2 CS #2: Dynamic Device Composition

Description. Mobile devices such as tablets and smart phones provide users with a high degree of mobility in accessing information such as emails, images and other resources. However, because of the limited size of their screens, these devices are not appropriate for accessing large quantities of resources. Because of this, a body of work (e.g. [36]) has explored the connection between small mobile devices and large situated horizontal displays like tabletops. In this case study, we demonstrate the ability to pair a mobile device (a tablet computer) to a tabletop display (Figure 6).

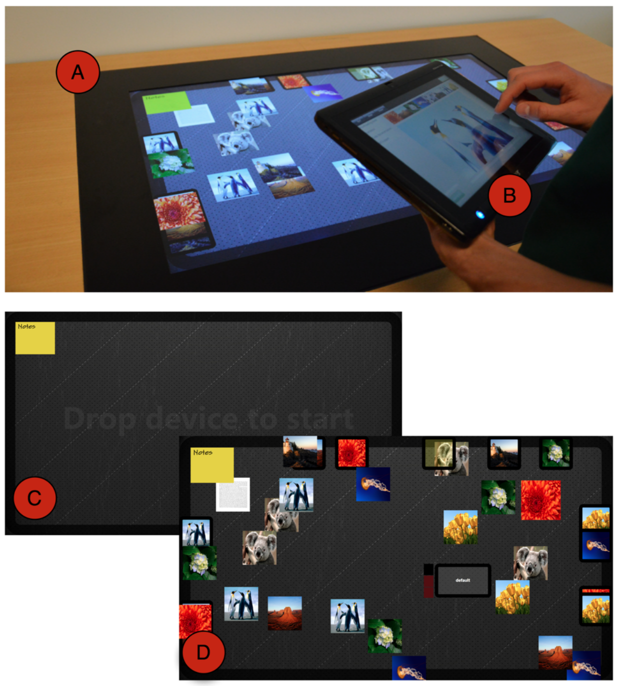


Figure 6: The public tabletop (A) connected to a tablet (B). The user walks up to an empty public interactive table (C) and simply places his device on top of the tabletop. The table will recognize the device and use the discovery to find and connect to the activity manager of the tablet. All resources related to the active activity on the mobile device are deployed on the interactive surface (D). When the user switches between activities, the interactive desk is repopulated with resources related to that activity. Users can utilize the desk to exchange, modify or manage files and resources. Additionally, the system allows the user to redirect input from the table, thereby providing remote control over the surface.

Using the Infrastructure. The main complexity and challenges in any application that supports multi-device composition on an interactive surface are (i) detecting and pairing with devices, (ii) file and resource synchronization and (iii) multi-user context (e.g. what resources belongs to what user). The interactive surface application was designed on top of the NooSphere *activity client* and *activity manager* (as seen in Table 1). However, in contrast to the co-Activity Manager application, the activity clients are dynamically started when a new device is detected. When the interactive surface detects a new device (using the vision system and static markers), it automatically launches the built-in discovery system to find a device with a running activity manager (in this case a tablet) of which the broadcast code matches the byte value of the detected tag. The broadcast code is taken from the device object that is initialized when the activity manager on the tablet is created. When the activity client on the surface computer pairs with the detected activity manager, all shared resource (images in this case) are automatically synchronized between both devices, and visualized on the surface. Because each loaded image is associated to a specific activity and its user, the system can distinguish the image set of each user. Additionally, rather than simply distributing files, the infrastructure provides a *Resources* object which encapsulates a file and annotates it with meta data. This data can be used by the application to do version control or check the association with multiple users and their activities. To support pointer redirect between different devices (allowing for remote control), the surface and tablet applications both implement a basic Context Service, which translates touch events from one device to another. The infrastructure adds the services to the running activity clients (e.g. as seen in the code sample in Figure 5) and automatically sets up and distributes the context information over a UDP multicast connection. Again, all complex multi-threaded network code, context modelling and device synchronization is hidden for the developer, allowing them to focus on the user interface and experiences. Because of all the supported services of the infrastructure, the total lines of code for the surface application is less than 800 lines.

User Experience. The synchronized activity state allows users to easily swap their set of resources on the interactive table. In a multi-user experience, this thus means that by simply selecting a different activity on the tablet, the user updates his part of the shared view. Because the infrastructure allows for easy addition of new resources, users can simply drag and drop resources from other users to their device. Again, very little *configuration work* is needed to exchange information or update the shared view. The pointer redirect can be enabled with a simple button click. Although some work on the side of the developer is required to support relative mapping, the master-slave negotiation and distribution of coordinates over the built-in UDP connection, creates an easy to use and transparent system for the end user. The user is thus given a very simple interface with advanced functionality hidden in NooSphere.

5.3 CS #3: SmartWard Research Prototyping

Description. In hospital patient wards, the whiteboard and patient record are two important artifacts to coordinate information concerning patients. In this case study, we demonstrate the first rapid prototype implementation of an ongoing research project in which we are constructing a distributed patient management system which supports multi-device configuration of patient cases as well as coordination through a number of automatic tracking, awareness and communication tools.

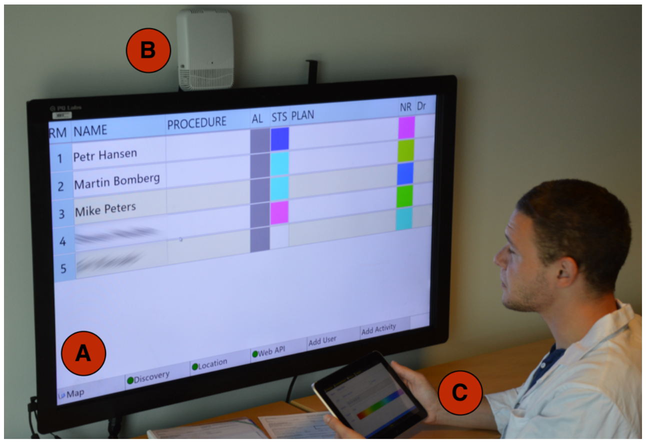


Figure 7: The SmartWard system consists of a large interactive whiteboard to display shared information on patients registered on the ward using RFID (A), a location tracker used to detect which patients are at the ward or in surgery (one node visible in B) and a tablet (C) used by doctors and nurses for more detailed information on the patient.

Using the Infrastructure. As illustrated in Table 1, this prototype uses all basic services provided by NooSphere. As in the other case studies, SmartWard uses *activity clients* and *activity managers* to synchronize activities, user information and resources across all devices. However, for this more domain specific application, we extended the infrastructure with a *WardNode* layer and specialized activity models (e.g. Patient, Nurses, Doctors,...). Figure 8 shows the code used by the application to (i) launch an activity system (manager or client) and (ii) and connect the distributed patient repository to a *ObservableCollection* that can be consumed by the UI. The WardNode is also connected to a Sonitor ultrasound location tracker (Figure 7 B) using NooSphere. The location tracker is a specialized context processor which runs in a separate service host managed by NooSphere. This means that the developer can simply enable the location tracking and use native events to deal with detections. Finally, the NooSphere event systems allows for the decoration of activities with custom tags and messages. In the SmartWard system, the patient records are attached to the custom patient model as activity-centric messages.

```
public BoardViewModel()
{
    WardNode = WardNode.StartWardNodeAsSystem(WebConfiguration.DefaultWebConfiguration);

    Patients = new ObservableCollection<PatientViewModel>();
    Patients.CollectionChanged += Patients_CollectionChanged;

    WardNode.PatientAdded += WardNode_PatientAdded;
    WardNode.PatientRemoved += WardNode_PatientRemoved;

    WardNode.PatientChanged += WardNode_PatientChanged;
    WardNode.Patients.ToList().ForEach(p =>
        Patients.Add(new PatientViewModel(p) {RoomNumber = _roomNumber++}));
}
```

Figure 8: The Wardnode class is a thin infrastructure extension which transforms NooSphere into a domain specific deployment. The code allows developers to create a distributed synchronized patient repository, which can be easily consumed by a MVVM application.

User Experience. To support coordination between clinicians, the infrastructure provides a patient activity for each active patient at the ward. This patient activity model is used to keep a strict synchronized whiteboard view but can also be used to share information with other clinicians. Since the information exchange services (such as writing nurse logs or updating the color state of a patient) are coupled directly to the patient activity model, there is no additional *configuration work* in locating the relevant contacts or starting an additional tool. Simply attaching information to the patient case will automatically distribute it to all relevant clinicians. The built-in support for location tracking provides clinicians with an easy to use and transparent search tool for other clinicians or artefacts (such as e.g. the patient record) at the ward.

6. DISCUSSION

Distributed interaction is a concept that has been around for many years, yet very few *infrastructures*, *toolkits* or *programming frameworks* that support the prototyping, development and deployment of these types of systems are actively in use. The central goal of NooSphere is to introduce a new *intermediate* [15] activity-centric infrastructure and programming framework that is aimed at providing a set of fundamental services required to design and deploy distributed interaction systems.

NooSphere uses activity as a *first class object* in an effort to reflect the intention of users in the modelling of information spaces that are spread over multiple devices, multiple locations and multiple people. Compared to traditional smart space and pervasive computing systems, this data model maps to the real physical tasks and activities people do in the information spaces provided by the infrastructure. This close match between the users' psychological interpretation of work and the digital aggregation of the resources required to perform this work, provides users with a stable mental model that has the capability to transcend the individual device. The model supports the notion of *actions* to structure work and resources and *user multiplicity* to allow multi-user access to the same activity model. Because of this activity model, the infrastructure allows for the creation of activity-centric coordination, configuration and communication tools that are part of the same *activity system*.

A central contribution of NooSphere is the aggregation of a cloud infrastructure, that is used for persistence and distribution of events, and a local dynamic distributed roomware infrastructure (similarly to COAST [37]). However, one of the core differences to prior smartspace systems is that NooSphere does not require specialized equipment or user interface frameworks but is usable with existing operating systems and UI toolkits. NooSphere thus encapsulates a number of complex services and systems into one activity-centric infrastructure, which is exposed through an API or standard REST interface. The main purpose of this approach is to provide a truly distributed and persistent platform that provides the ability to interconnect systems distributed over different locations all over the world. Combining a dynamic smart room environment with the persistence of an integrated cloud platform opens up possibilities for new collaborative setups distributed over multiple locations. This simplification of interconnections between distributed services or "'*Power in combination*" [31] results in a new design and prototyping platform. By providing a standard architecture and model for activity-centric computing, we provide developers with a framework to built interconnectable tools.

Prototyping complex activity-centric distributed system is *easier* as a developer is provided with a set of basic services which are *flexible*, easy to set up and *transparent*. All network code, discovery mechanisms, file and activity synchronization, and context handling are abstracted into the infrastructure and presented to the developer as basic Mono `C#` objects and delegates. Because of this, prototyping and designing distributed user interfaces is significantly *faster* as it requires less lines of code (to debug). Because of the abstract model iterative changes to the design (e.g. induced by user-centric design) do not require the re-engineering of (parts of) the infrastructure. The architecture of the infrastructure is extensible as controllers and services can be added, allowing for modifications, extensions and integration with other platforms. The infrastructure is designed to support a broad range of technical setups ranging from traditional local client-server-cloud (e.g. Case Study #1) and peer to peer (e.g. Case Study #2) setups to large complex cloud-based hybrid setups (e.g. Case Study #3). Because of the two-layered architecture and component based design, the infrastructure is *scalable* and *reusable* for complex distributed applications.

The infrastructure currently also has a number of *challenges* and *limitations*. Some services, such as the context processor or discovery mechanism, provided in the local activity system are not usable in the cloud. Although the infrastructure allows for messaging between activity systems using the cloud event controller, this approach is practically not feasible for e.g. discovery or high bandwidth real-time context data. The current two-tier architecture of NooSphere is grounded in the design rational that any device that is part of the activity system is connected to a local network. This implies that a local device can always be setup as a local activity manager, thus providing a node with the necessary services. However, there are a number of use case (e.g. using smartphones on a 3G network) where these services can currently not be provided. E.g. if the tablet from Case Study #1 would be connected to the activity cloud over 3G, the local system would not be able to detect it. Although the device would be in the same room, the event distribution would be done over the cloud, not the local system.

Because of the high level of abstraction of the activity model and infrastructure design, some use cases require a thin infrastructure layer on top of the standard NooSphere API. E.g. in Case Study #3, a domain specific layer was constructed to encapsulate some of the dynamic ad-hoc node creation as well as an implementation of the location tracker. This thin layer is not a formal requirement as the same functionality can be achieved on the bare framework code. However, adding this thin layer can facilitate development and help to manage the complexity of more advanced setups. Although NooSphere provides developers with a number of `C#` objects and event and a REST API for other programming environments, there is currently still an *integration problem*. Because the infrastructure is primarily focused on data, event and context distribution, the development and integration of these concepts into the user interface is still left in the hands of the developers. Although NooSphere greatly reduces the amount of work on the distribution part, building activity-centric user interfaces (such as [4, 13, 20, 29, 40]) is still a challenging task. A next step could thus be to extend the API of the infrastructure to deeply integrate with existing operating systems and widely used systems and tools, to provide an even broader development platform, or *activity-based toolkit*.

7. CONCLUSION

In this paper we introduced NooSphere, an activity-centric service-based infrastructure for the prototyping of distributed interaction systems. We described the motivation, architecture and components, and presented three example applications build using NooSphere. We are currently using the infrastructure for different research projects aiming at deploying multi-device computing support in hospitals, interactive desks for knowledge workers, and distributed collaboration in global software development.

8. ACKNOWLEDGEMENTS

This work was supported by the EU Marie Curie Network iCareNet under grant number 264738. Thanks to Steven Jeuris for his suggestions and feedback on early drafts of this paper.

9. REFERENCES

[1] Arthur, R., and Olsen, Jr., D. R. Xice windowing toolkit: Seamless display annexation. *ACM Transactions on Computer-Human Interaction 18* (August 2011), 14:1–14:46.

[2] Arthur, R., and Olsen, Jr., D. R. Window brokers: Collaborative display space control. *ACM Transactions on Computer-Human Interaction 19*, 3 (Oct. 2012), 17:1–17:21.

[3] Baldauf, M., Dustdar, S., and Rosenberg, F. A survey on context-aware systems. *International Journal of Ad Hoc and Ubiquitous Computing 2*, 4 (2007), 263–277.

[4] Bardram, J., Bunde-Pedersen, J., and Soegaard, M. Support for activity-based computing in a personal computing operating system. In *Proc. of CHI '06*, 211–220.

[5] Bardram, J., Gueddana, S., Houben, S., and Nielsen, S. Reticularspaces: Activity-based computing support for physically distributed and collaborative smart spaces. In *Proc. of CHI 2012*, ACM (2012).

[6] Bardram, J. E. The Java Context Awareness Framework (JCAF) - A Service Infrastructure and Programming Framework for Context-Aware Applications. *Pervasive Computing* (2005), 98–115.

[7] Bardram, J. E., Bunde-Pedersen, J., Doryab, A., and Sørensen, S. Clinical surfaces - activity-based computing for distributed multi-display environments in hospitals. In *Proc. of INTERACT '09*, 704–717.

[8] Biehl, J. T., and Bailey, B. P. Aris: an interface for application relocation in an interactive space. In *Proc. of GI 2004*, 107–116.

[9] Biehl, J. T., Baker, W. T., Bailey, B. P., Tan, D. S., Inkpen, K. M., and Czerwinski, M. Impromptu: a new interaction framework for supporting collaboration in multiple display environments and its field evaluation for co-located software development. In *Proc. of CHI '08*, 939–948.

[10] Black, J. P., Segmuller, W., Cohen, N., Leiba, B., Misra, A., Ebling, M. R., and Stern, E. Pervasive computing in health care: Smart spaces and enterprise information systems. In *Proc. of the MobiSys 2004*,

[11] Consolvo, S., McDonald, D. W., Toscos, T., Chen, M. Y., and Froehlich, J. e. a. Activity sensing in the wild: a field trial of ubifit garden. In *Proc. of CHI '08*, 1797–1806.

[12] Dey, A., Abowd, G., and Salber, D. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Human–Computer Interaction 16*, 2-4 (2001), 97–166.

[13] Dragunov, A. N., Dietterich, T. G., Johnsrude, K., McLaughlin, M., Li, L., and Herlocker, J. L. Tasktracer: a desktop environment to support multi-tasking knowledge workers. In *Proc. of IUI '05*, 75–82.

[14] Edwards, W. K., Bellotti, V., Dey, A. K., and Newman, M. W. The challenges of user-centered design and evaluation for infrastructure. In *Proc. of CHI '03*, 297–304.

[15] Edwards, W. K., Newman, M. W., and Poole, E. S. The infrastructure problem in hci. In *Proc. of CHI '10*, 423–432.

[16] Garlan, D., Siewiorek, D. P., Smailagic, A., and Steenkiste, P. Project Aura: Toward Distraction-Free Pervasive Computing. *IEEE Pervasive Computing 1*, 2 (April 2002), 22–31.

[17] Gjerlufsen, T., Klokmose, C. N., Eagan, J., Pillias, C., and Beaudouin-Lafon, M. Shared substance: developing flexible multi-surface applications. In *Proc. of CHI '11*, 3383–3392.

[18] Haibin Zhu, M. Z. Role-based collaboration and its kernel mechanisms. In *Proc. of the 3rd IEEE International Workshop on Distributed Intelligent Systems* (2006).

[19] Hong, J.-y., Suh, E.-h., and Kim, S.-j. Context-aware systems: A literature review and classification. *Expert Syst. Appl. 36*, 4 (May 2009), 8509–8522.

[20] Houben, S., Bardram, J., Vermeulen, J., Luyten, K., and Coninx, K. Activity-centric support for ad hoc knowledge work - a case study of co-activity manager. In *Proc. of CHI '13*, 2263–2272.

[21] Houben, S., Esbensen, M., and Bardram, J. A situated model and architecture for distributed activity-based computing. In *Proc. of MODIQUITOUS '12*, 9–13.

[22] Johanson, B., Fox, A., and Winograd, T. The interactive workspaces project: Experiences with ubiquitous computing rooms. *IEEE Pervasive Computing 1* (April 2002), 67–74.

[23] Johanson, B., Hutchins, G., Winograd, T., and Stone, M. Pointright: experience with flexible input redirection in interactive workspaces. In *Proc. of UIST 2002*, 227–234.

[24] Kumar, M., and Shirazi, B. A. Pico: A middleware framework for pervasive computing. *IEEE Pervasive Computing 2* (2003), 72–79.

[25] Lenk, A., Klems, M., Nimis, J., Tai, S., and Sandholm, T. What's inside the cloud? an architectural map of the cloud landscape. In *Proc. of the ICSE Workshop on Software Engineering Challenges of Cloud Computing 2009*, 23–31.

[26] Li, Y., and Landay, J. A. Activity-based prototyping of ubicomp applications for long-lived, everyday human activities. In *Proc of CHI '08*, 1303–1312.

[27] Lu, K., Nussbaum, D., and Sack, J.-R. Globecon: A scalable framework for context aware computing. In *Smart Sensing and Context*, G. Kortuem, J. Finney, R. Lea, and V. Sundramoorthy, Eds., Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2007.

[28] Moran, T. P., and Zhai, S. *Beyond the desktop in seven dimensions*. The MIT Press, 2007, ch. 11, 335–354.

[29] Muller, M. J., Geyer, W., Brownholtz, B., Wilcox, E., and Millen, D. R. One-hundred days in an activity-centric collaboration environment based on shared objects. In *Proc. of CHI '04*, 375–382.

[30] Norman, D. A. *The invisible computer: why good products can fail, the personal computer is so complex, and information appliances are the solution*. The MIT press, 1999.

[31] Olsen, Jr., D. R. Evaluating user interface systems research. In *Proc. of UIST '07*, 251–258.

[32] Padovitz, A., Loke, S. W., and Zaslavsky, A. The ecora framework: A hybrid architecture for context-oriented pervasive computing. *Pervasive Mob. Comput. 4*, 2 (Apr. 2008), 182–215.

[33] Pering, T., Want, R., Rosario, B., Sud, S., and Lyons, K. Enabling pervasive collaboration with platform composition. In *Pervasive Computing*, H. Tokuda, M. Beigl, A. Friday, A. Brush, and Y. Tobe, Eds., vol. 5538 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2009, 184–201.

[34] Reiff-Marganiec, S., Truong, H.-L., Casella, G., Dorn, C., Dustdar, S., and Moretzky, S. The incontext pervasive collaboration services architecture. In *Proceedings of ServiceWave '08*, Springer-Verlag (2008), 134–146.

[35] Román, M., Hess, C., Cerqueira, R., Ranganathan, A., Campbell, R. H., and Nahrstedt, K. Gaia: a middleware platform for active spaces. *SIGMOBILE Mob. Comput. Commun. Rev. 6*, 4 (Oct. 2002), 65–67.

[36] Shen, C. From clicks to touches: enabling face-to-face shared social interface on multi-touch tabletops. In *Online Communities and Social Computing*, Springer (2007), 169–175.

[37] Streitz, N. A., Geißler, J., Holmer, T., Konomi, S., Müller-Tomfelde, C., Reischl, W., Rexroth, P., Seitz, P., and Steinmetz, R. i-land: an interactive landscape for creativity and innovation. In *Proc. of CHI 1999*, 120–127.

[38] Thompson, M. S., Plymale, W. O., Hager, C. T., Henderson, K., Midkiff, S. F., Dasilva, L. A., Davis, N. J., and Park, J. S. Ad-hoc networking support for pervasive collaboration. In *in Adjunct Proc. of UbiComp04*,

[39] Toninelli, A., Montanari, R., Kagal, L., and Lassila, O. A semantic context-aware access control framework for secure collaborations in pervasive computing environments. In *International Semantic Web Conference* (2006), 473–486.

[40] Voida, S., Mynatt, E. D., and Edwards, W. K. Re-framing the desktop interface around the activities of knowledge work. In *Proc. of UIST '08*, 211–220.

[41] Weiser, M. The computer for the 21st century. *Scientific American 265*, 3 (Sept. 1991), 66–75.