2017/18 Mini-Project

Osmose: An Architecture for Interspatial Networking

Final Report

Anil Madhavapeddy, Gemma Gordon, KC Sivaramakrishnan
avsm2@cl.cam.ac.uk, gg417@cl.cam.ac.uk, sk826@cam.ac.uk
Cambridge Computer Laboratory

Abstract

Digital infrastructure in modern urban environments is currently very Internet-centric, and involves transmitting data to physically remote environments. The cost for this is data insecurity, high response latency and unpredictable reliability of services. In this research, we design a software architecture that inverts the current model by building an operating system designed to securely connect physical spaces with extremely low latency, high bandwidth local-area computation capabilities and service discovery. Our early prototype design Osmose is based on unikernels and a distributed Irmin store.

Research Question

In his classic essay on ``Computing for the 21st Century'', Mark Weiser observed that “the most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it”. Since then, there have been tremendous advances in mobile and sensing technologies, and there is an ongoing rapid deployment of “smart" digital infrastructure that augments the physical environment. Consider the following scenario for a next-generation building:

“A group of people are meeting in an co-working space to discuss a project they are working on together. They are the first to arrive at the building on a cold winter morning, and are directed to a meeting room automatically by audio directions individually projected to them. As they walk into the room the lights are already on and the heating has been preset to a comfortable level. They wave to a wall where a display appears, and the building recognises the gesture and projects a shared folder of the project. It begins to record and encrypt the conversation with a group secret keyed to the participants. One of the participants receives a tight-beam audio notification informing them that they are needed elsewhere briefly. When they leave the room the shared recording is immediately paused within milliseconds to allow other participants to casually chat among themselves. Subsequently, the encrypted recording is transcribed, and made available in the datastores of each participant under the terms of their shared disclosure agreement for their project.”

This scenario illustrates the distance between Weiser's 1999 vision of ubiquitous computing and where we currently are almost two decades on. The above narrative features *no wearables or mobiles* among the human participants; instead of an array of individual smart phones, watches and laptops, the situated environment of the building is providing secure, shared, multi-tenant infrastructure for audio and visual communications among the human participants. This is analogous to the shift from individual computers to multi-tenant cloud infrastructure in the past decade, but applied to the sharing of physical places and devices.

Another aspect to consider in the scenario is the seamless, low-latency nature of the interactions that were described between humans and electronic devices. When a human takes an action the environment is able to react within milliseconds to take action immediately, rather than delaying for seconds and causing the awkward delays that we have become used to with modern Internet-connected mobile devices. By providing immediacy, the technology can complement social interactions rather than interrupting them.

Encouragingly, much of the hardware required to physically assemble the described building is available off the shelf. E-ink display wallpaper, parametric audio speakers for directional broadcast, gesture recognition and smart lighting and heating are all available or relatively easy to construct. The missing link is the foundational software that manages, coordinates and secures the distributed hardware -- the operating system for digitally connecting physical spaces together.

Our research question then, is how to design a next generation software operating system that is suitable for powering the buildings that we live in.

Methodology

We begin to bridge this gap between available hardware and missing software by constructing an architectural model for *interspatial networking;* an operating system for the dense, interconnected and shared urban spaces that most humans live in. It aims to shift us away from the wide-area, mobile-oriented devices that are permeating early deployments of smart infrastructure, and towards a sustainable digital model that is far more similar to a conventional utility such as electricity or gas. Our system aims to make it far simpler and more secure to introduce, manage and rely on digital devices during day-to-day life in urban environments.

There is also an exciting new generation of upcoming applications that demand vast amounts of bandwidth and low-latency responses. Augmented and virtual reality, environmental e-ink displays, parametric directional sound, and robotic appliances simply do not work well if the latencies of interactions with them rise above a certain point. We propose an operating software architecture aimed at fully supporting these new applications on modern hardware platforms.

Given that the hardware is all available, why is the above scenario difficult to implement? The answer lies in the traditional operating systems and network architecture that power the current generation of smart devices. Because of the rise of cloud computing, they are typically built around communication to centralised Internet services. For example, consider what happens when we speak into an Apple Watch in order to retrieve some information. For this to work, the watch must be connected to a mobile phone, which in turn needs to establish a wireless connection to the Internet, where Apple voice recognition services will dispatch a query to the Google search engine. If any one of the services in this chain breaks (for example, the common case of the phone signal being ``trapped'' by a wifi authentication page), then the user experience is broken. Even when it does work, it can take seconds to respond to the voice query, and with very variable latency. Once the response comes through, handoff to other devices is also difficult unless they are owned by the user.

Reliability: Services deployed in physical environments need to work all the time, and be locally debuggable when they do fail. How do we shift from a light switch that ``almost works'' after being pressed several times to seamless voice or gesture-driven services that are always tuned and available in a given environment? They also need to work independently of Internet connectivity, so that every building can be an island of digital services even when offline.
Security: The amount of sensitive data being captured in these environments is tremendous, and much of it should not leave the confines of the physical space without explicit permission from all parties involved. This is extremely difficult to police given the amount of Internet-wide coordination used in existing devices, but can be fixed if the local environment provides a structured mechanism for handling such storage securely with respect to local environmental policies.
Latency: Interactive services require response times beneath the uncanny valley of human perception. For the scenario above to really feel seamless, we need a new ``latency first'' application architecture that makes data and computation capacity available physically near the human users, with scheduling tolerances for responses in the milliseconds rather than seconds.

Solving these problems is difficult to do piecemeal across individual parts of the software stack, since they are currently general-purpose and loosely coupled. A typical IoT device might run its own copy of the Linux kernel, with an embedded userspace, a VPN into a centralised management server

operated by the vendor for updates, and a mobile application for the user to manage it. There is little synergy or dependence on shared infrastructure to assist with the process.

Discussion

When dealing with physical devices, the latency of response to external events is paramount. Safety critical systems often require “hard realtime” operation, or more often attempt to provide soft guarantees about when they respond. As we move further up the stack to modern mobile applications, there are no such guarantees. Pressing a button on an iOS or Android device goes through many layers of scheduling and network connectivity before a response is generated.

Our interspatial architecture needs to allow us to deploy “latency-first apps” into physical environments; ones that are designed to run on local devices with response budgets in the milliseconds, and with minimal external connectivity needs. This implies that there is simply no time for doing conventional operations such as network scanning or service discovery in serial -- by the time the user has asked a question, the physical environment should already have the appropriate resources established. We thus need to rearrange the programming models around building interspatial applications to make them suitable for such an environment. We adopt the following design principles:

Incremental Networking: While it is currently possible to spin up services on demand within milliseconds, it is difficult to transmit a response for complex clustered services within a time budget of a few milliseconds. A device that is starting ``cold'' will need to establish a network connection to the local node, most commonly via TCP and subsequently a secure transport via TLS. The number of hops required to establish a secure connection have been recognised as a problem on the wider Internet, and new low-latency protocols such as QUIC are being deployed in browsers. However, even these newer protocols only solve part of the problem, since they do not integrate closely with the application layer. Once the connection has been established, we still need to authenticate the user(s), negotiate security keys, perhaps perform version negotiation between devices, and usually interrupt the user at an inconvenient time for a software security update.

The reason that every device has to currently do all this work is partly due to the end-to-end principle that guides the design of Internet protocols. Every IP node is a “dumb” router that knows how to forward protocol packets, but lacks higher-level application information. Existing system interfaces (such as the BSD sockets API, or DNS resolution) implement this network stack, which is in turn baked into existing application logic. Each device needs to repetitively perform the same actions to establish connectivity, with the surrounding network environment being unable to assist.

An alternative interface in a physical environment is to move away from just-in-time connectivity towards a model of establishing connectivity incrementally as the opportunity arises. When sufficient information from the environment arises to establish a partial connection (for example, upon entering a building but before making any service requests) the application needs to perform the operations that it can do so then, such as negotiating security keys or ensuring that software versions match. When a service request does finally happen, the previously derived connection information can be used to perform a single hop. Incremental connectivity requires fundamental programming interface changes in order to combine information across the traditional operating system and application stacks, and for us to move away from the venerable sockets API.

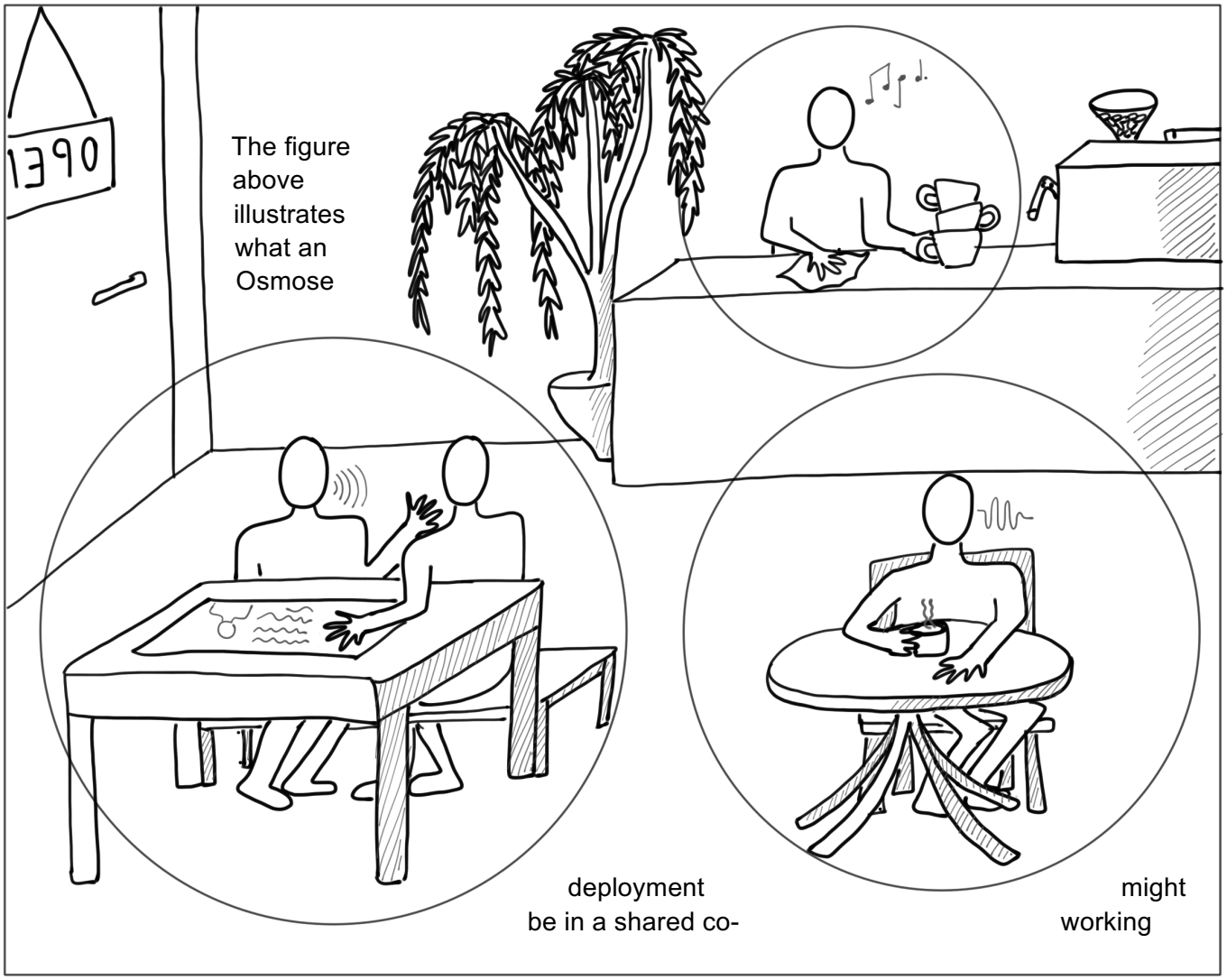
Spatial Interfaces: Incremental networking lets an individual network connection to a node be established quickly. In a physical environment it is also necessary to mesh wired sensors (such as

motion detectors or cameras) with mobile ones (humans carrying location beacons). The wired systems need to be establishing connectivity with services required by applications, and the wireless ones need to be searching for nearby network nodes via Zigbee or Bluetooth. This connectivity mesh is necessary to ensure that all of the distributed components that comprise a physical environment can all communicate with predictable and very low latency.

Internet protocols such as TCP have not traditionally not handled connection handoff well -- extensions such as multipath TCP are effective but have struggled with extending existing interfaces such as the sockets API with the new routing semantics available to applications.

For our interspatial applications, a building needs to be able to manage all the local resources (include bandwidth or storage) just as it does so with other utilities such as electricity. In return, individual applications do not need to manage their own networking, storage and authentication needs as they are provided by the surrounding environment. This requires a change in the traditional programming model to allow applications to multiplex their connectivity needs around a networking environment that is multipath and dynamic as humans move around the physical space.

Native Hardware: Applications are traditionally built for a specific device profile in mind -- for example, a mobile phone or a desktop application. The proliferation of multiple display form factors has resulted in a new mode of ``responsive'' design that can adapt an interface to multiple resolutions and sizes. With interspatial applications, we wish to eliminate the need for importing wearable and mobile devices into a building as the sole mechanism of interaction with a user. Hardware present in the environment such as parametric speakers for 3D positional audio or wallpaper projections should be able to be used by local users, rather than being constrained to the single form factor mobile devices (e.g. a watch or a mobile phone) that we carry around. For this to work, interspatial applications needs to be designed to have responsive user interfaces, but also a suitable trust model to let users establish secure connections to environmental hardware that is not directly owned by them



space. Instead of the usual crowded array of laptops, mobile phones and headphones, the users benefit from use of the hardware present in the situated environment. The two users on the bottom left can collaborate using a touchscreen built into the table, and their conversation transcribed and encrypted via local microphones that can filter out noise from other tables due to their proximity to the user. Meanwhile, the lone worker on the bottom right can listen to an encrypted podcast via parametric audio speakers that project the sound directly from the ceiling, without a need for the user to hook up headphones and a mobile device. Finally, the barista at the top right can work while listening to their music selection with personalised notifications, and also pause it as soon as a customer requires the barista's attention.

Behind the scenes, Osmose is providing important operating system services to this scenario: each of the users has been authenticated upon entry to the building and could install their interspatial applications using the local wireless network. They each have their own personal policies (e.g. audio sources and conversation transcription) that have been installed into the building. Each of the users need to be tracked by the building with low latency to ensure that they can immediately interact with the devices in front of them, while also enforcing isolation of their data across the shared hardware in the building such as the microphones and parametric audio speakers. As they purchased their caffeinated beverages and took their seats, other services such as payments could also be potentially performed securely using this infrastructure.

Conclusion

Our Osmose prototype design brings the traditional benefits of an operating system to the distributed array of hardware that comprises a physical building --- resource scheduling, hardware isolation and a userlevel programming interface. The programming model is aimed at building real-time, interactive interfaces that can do complex data processing local to the user, without being forced to ship data remotely to the cloud.

Any operating system is only worth the applications that run on it. While our application interfaces are intentionally not backwards compatible with existing frameworks to give us room to experiment, their underlying programming model is a familiar one to those who program cloud-based infrastructure. We are hoping that the micro service-like application model combined with automated deployments on embedded devices will be compelling to developers.

Our next steps are to create a fuller specification of the design outlined in this position paper and evaluate realistic next-generation interactive applications on this platform, and to build a real physical implementation of our design in a building environment. All of the source code of the constituent components discussed here such as MirageOS and Irmin are available under a BSD-style license from https://github.com/mirage.

Related and Further Work

An expanded version of this work was published at IEEE HotPOST 18 (https://www.semanticscholar.org/paper/An-Architecture-for-Interspatial-Communication-Madhavapeddy-Sivaramakrishnan/d5f1b82f43c51b0ba5aeb7132457a58b54b8308d) with more technical details of the architecture.

Acknowledgements

This project was supported by a mini-projects award from the Centre for Digital Built Britain, under InnovateUK grant number 90066