Unified Operating Superstructure

and

the Abstraction-based Morphable Interface

(Continuously updated)

https://ultranet.org

Abstract: The unified operating superstructure (UOS) allows developers to create platform-independent applications that are run as easily as the opening of a Web page, but with the full power of a local system and without the issues typical of today's cross-platform development. The technology aims to take the high performance and rich UI capabilities of local platforms while inheriting platform-independent network nature, safety, and ease of use from the Web platform.

The Abstraction-based Morphable Interface (AMI) introduces an open UI technology for UOS applications that defines a virtual user interface that can morph to adapt to any particular physical device. Many and various Web applications can utilize the power of a AMI to provide a rich and portable user-interface experience that would be unattainable with classic HTML-based technologies. Applications built with AMI are not limited by classic 2D UIs, but are also capable of rendering any VR visual interfaces, which is important for support existing and future VR and AR gears.

Contents

Glossary

GUI Graphical user interface: a form of user interface that allows users to interact with electronic devices through graphical icons and visual indicators such as secondary notation

HTML Hypertext Markup Language: the standard markup language for creating Web pages and Web applications

HTTP Hypertext Transfer Protocol: an application protocol for distributed, collaborative, hypermedia information systems. HTTP is the foundation of data communication for the World Wide Web, wherein hypertext documents include hyperlinks to other resources that the user can easily access.

OS Operating system: system software that manages computer hardware and software resources and provides common services for computer programs

VR Virtual reality: an experience taking place within simulated and immersive environments that can be similar to, or completely different from, the real world

The Web UI Paradigm

Every site on the Web uses HTML to generate its pages. HTML is a static text markup of page elements, which was developed as a language that allows the creation of rich-text documents. A rich-text document is a text that can also contain various media elements such as images, animations, links, tables, and other items. JavaScript language is used to bring some dynamics to the static nature of HTML pages. Finally, the language of Cascading Style Sheets (CSS) makes Web design and markup considerably easier.

These are three major technologies that all websites use, and these technologies are perfect for sites that represent themselves mainly as linked rich-text documents. An additional benefit of text-based Web technology is the capability of being intrinsically indexable by search engines. As the Web has developed and grown, though, something more flexible and faster than HTML has become a critical requirement for many websites. If we take a close look at these websites, it becomes clear that they are in fact more applications than sites, and that their requirements differ from those of websites *per se*. They need a complex and responsive GUI, they need to work at the highest possible level of performance, and they do not need to be indexable by search engines.

Here are some examples of these Web applications:

- Social networks and other sites that provide their content for authenticated users only
- Corporate, public, and other portals
- Video, photo, and other media hosting sites
- Site admin. and personal sections
- Email clients
- Games
- Development tools, task managers, and source controls
- Exchanges, online banking, accounting, and POS
- Services control panels such as Web hosting panels
- Online office tools
- Maps
- Streaming services
- Various online tools such as converters and editors
- Many, many others.

Several technologies provide solutions, the best known of which is Flash. While Flash allows the creation of things that are impossible under pure HTML and JavaScript, this is a proprietary product, and still somewhat alien to the Web. The Internet still needs some technology that will deliver the power and flexibility needed for Web applications, but security requirements hinder progress in this area. Existing Web technologies do not allow the running of native code and by design do not provide access to the resources of the local OS. These restrictions help to prevent almost all kinds of malicious activity by website code, and they make Web surfing much safer than installing software locally.

Local Platforms UI Paradigm

By the term *local platforms,* we mean locally installed desktop or mobile operating systems such as Windows, Linux, macOS, iOS, Android, and the like. *Local or installable applications*, in turn, are software programs installed and run on local platforms. Software vendors want their products to run on as many platforms as possible. However, cross-platform software development is difficult, as platform differences and the look and feel of different GUIs frequently create a dilemma:

EITHER

a quick and cheap development using some cross-platform framework with bad user experience,

OR

a long-term and expensive, mostly separate, development for each platform with the best user experience.

On the other hand, though, are there really any significant differences between the local operating systems available on the market? After all, all of the systems come with a very similar set of functions – multitasking, 2D-window GUI, security, networking, and some others – so it is clear that cross-platform development difficulties caused by inter-platform differences are not inevitable. If we could introduce a standard for cross-platform APIs and a unified OS-independent GUI standard, then we would be free from problems caused by inter-OS specifics and differences.

Another problem with current platforms is that their GUIs do not fit properly on existing and future 3D-display devices, and it would be nearly impossible to effectively adapt legacy 2D-window user interfaces to the 3D environment provided by 3D gears. Furthermore, the classic GUI has some problems with utilizing the screen space of wide and ultra-wide display monitors: Maximizing makes windows too big, and finding a good solution for this will very likely require rewriting an application's code.

Existing solutions

Several existing technologies try to address the major issues of local and Web applications. Table 1 below gives a comparison of some of their pros and cons.

Table 1. Pros and Cons of Existing Solutions

	HTML5	Flash (Adobe)	AIR (Adobe)	ActiveX (Microsoft)	Silverlight (Microsoft)
Industry standard	Yes	No	No	No	No
Open source	Yes	No	No	No	No
Cross-platform support	Yes	Partial	Partial	No	No
Local system API	No	No	Partial	Yes	Partial
UI integration (mostly)	Inside a browser	Inside a browser	Local OS UI	Inside a browser	Inside a browser
Performance	Low	Moderate	Moderate	High	Moderate
Maximum acceptable level of UI complexity	Low	Moderate	Moderate	Moderate	Moderate
Malware resistance	High	Low	High	Low	High
VR support	No	No	No	No	No
Deprecated	No	Yes	No	Yes	Yes

Requirements

Given the problems described above, let us summarize all the requirements of a new technology:

- A platform that allows the building of OS-independent applications that are accessible in a similar way to Web pages using commonly known URL navigation, but appear and function like conventional local applications
- An application lifecycle that should not include installation and uninstallation stages
- The possibility of replacing application-like HTML websites with a technology that brings the power of installable applications
- Platform-independence
- A unified UI that is able to adapt a single platform-independent user environment to desktop, mobile, and VR environments, and to leverage the full power of local hardware. It should support any desktop and mobile devices as well as modern and future VR gears.
- An open technology to avoid any patent-infringement cases; a profit-oriented evolution strategy; and promotion of proprietary software.

Unified Operating Superstructure

The UOS provides safe execution and exposes the unified API for UNA applications. A virtualized and isolated execution environment is a concept that minimizes the impact of malicious software or incorrect behavior of a running application. This security measure, also known as a “sandbox”, is similar to how applications work on iOS and Android. In this environment, each application could be in one of the following three isolation states:

- **Isolated**: Execution is allowed within a virtual environment.
- **System**: Execution is allowed, and virtualization is disabled so the application can access any resources it needs – even the data of other applications.
- **Blocked**: A low trust level is received from the DMS, or antimalware found its code suspicious and had this confirmed by the user, so prevented the application from executing.

The isolated application only has access to the virtual file system and to a virtualized system API. This is the last line of defense and it is needed to prevent undetected malicious code from harming a user system or stealing sensitive data. It works similar to sandboxes in browsers and mobile OSs.

The Unified Operating Superstructure provides access to a host OS via an open-standard platform-independent API framework. Both native and LLVM APIs are provided. Platform-independent applications are distributed in the form of LLVM code. Particular LLVM implementation is subject to further research. CIL and WASM are currently major candidates. Applications developers can choose the type of code they want to distribute. The best way is to publish LLVM code, which can run on any platform. However, if a developer wants to provide the highest performance, then s/he can create an application using a native code approach and publish binaries for all major platforms. In the latter case, when the application is requested to run, appropriate binaries will be downloaded and used according to the type of local platform.

There is no longer an installation process – an application is started once all required components are downloaded. To speed up the loading of the applications, its architecture should be designed to consist of three levels of components:

Core When the application is initially requested, only the core components are downloaded, and the default configuration is used to launch the application.

Deferred After the application is started, the deferred loading components are downloaded in the background.

Optional Finally, when the user needs some additional features, s/he can ask the system to download the corresponding extension components.

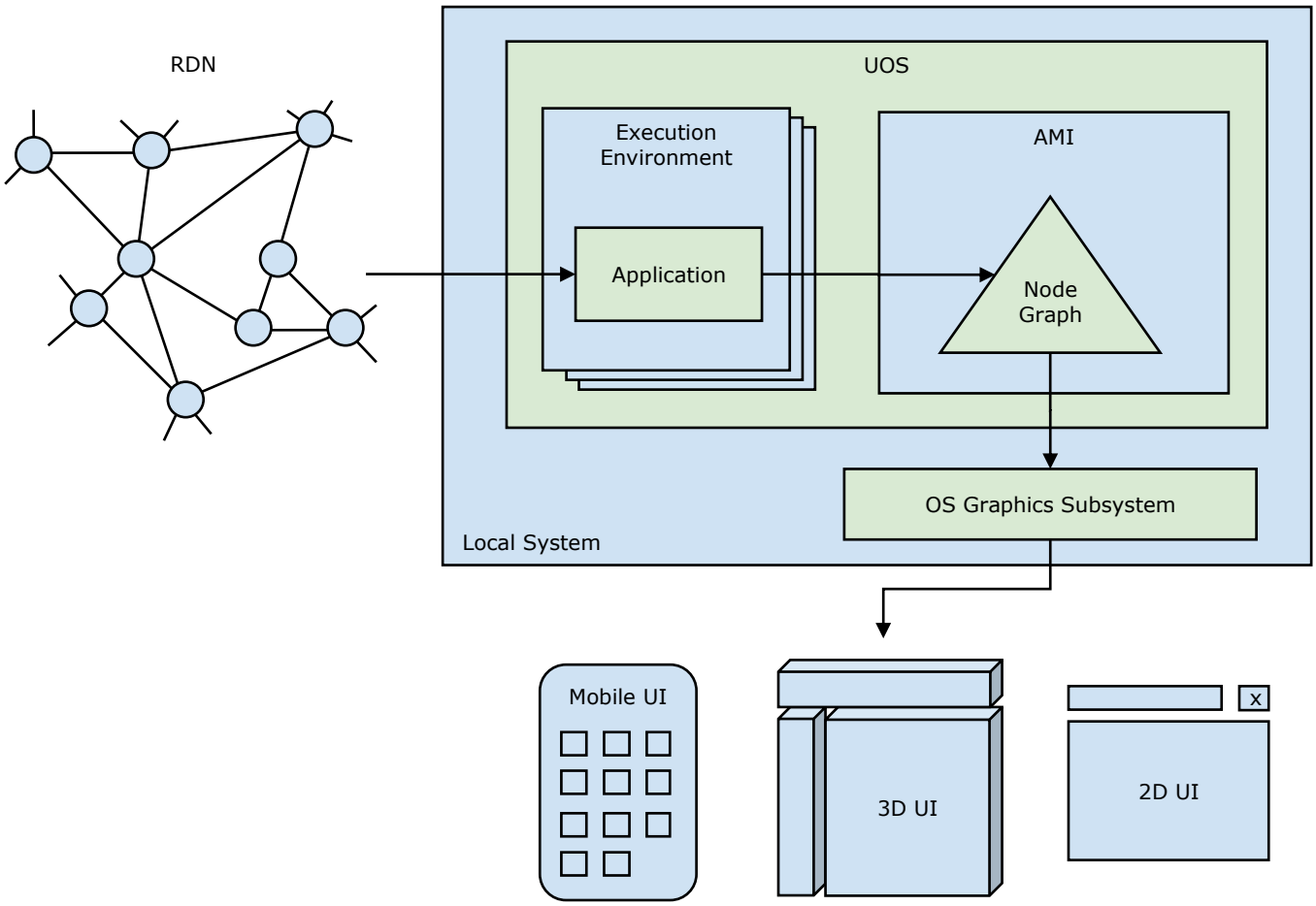
The Abstraction-based Morphable Interface

The major ideas behind the AMI are as follows:

- The user does not run programs; instead, s/he opens applications from the Internet and builds a personal environment.
- The environment is independent of any particular physical device.
- The technology can render any user interfaces, not only regular windows or Web pages but also any possible 3D UI, in a single visual environment.
- Ordinary display devices are supported, and the capabilities of modern and future VR gears are utilized.
- Any object has a unique global address in the form of a URL to support referencing.

The concept is based on the Aximion project. The mission of Aximion is to take the next step in UI evolution by providing a new visual experience that inherits the important features of the classic windows GUI and the Web-like nature of Aximion objects and adds the capabilities of modern and future display devices and VR gears. The special open standard will be introduced to guarantee the same look and feel across all OS platforms. The development of this standard is still at a very early stage but it is going to be like some hybrid of HTML and OpenGL.

Figure 1. The place of AMI



Unlike any of the usual OSs, in the AMI, all root UI nodes have a global persistent unique address in the form of a URL. This gives UI objects capabilities intrinsic to Web resources, such as links, history, etc. and makes their life cycle different from that used in ordinary operating systems. With such a capability, it is now possible to globally refer to any object using URL semantics. Whenever a user creates something, it always gets a predefined or random unique address assigned to it, and all data related to that instance is kept until at least one reference to this object exists. This allows the user to link to any objects exposed by the application in the same way as s/he does for files, Web pages, and other resources.

At the shell level, several UI concepts have been introduced in Aximion to help users personalize and organize their environments.

The Fields

The AMI rethinks the idea of the desktop. In the AMI, desktops have evolved into *fields*. The field is a place where different aspects of a single project/context/interest can be grouped into a single UI entity.

Figure 2. Present-Day Application-Context Paradigm

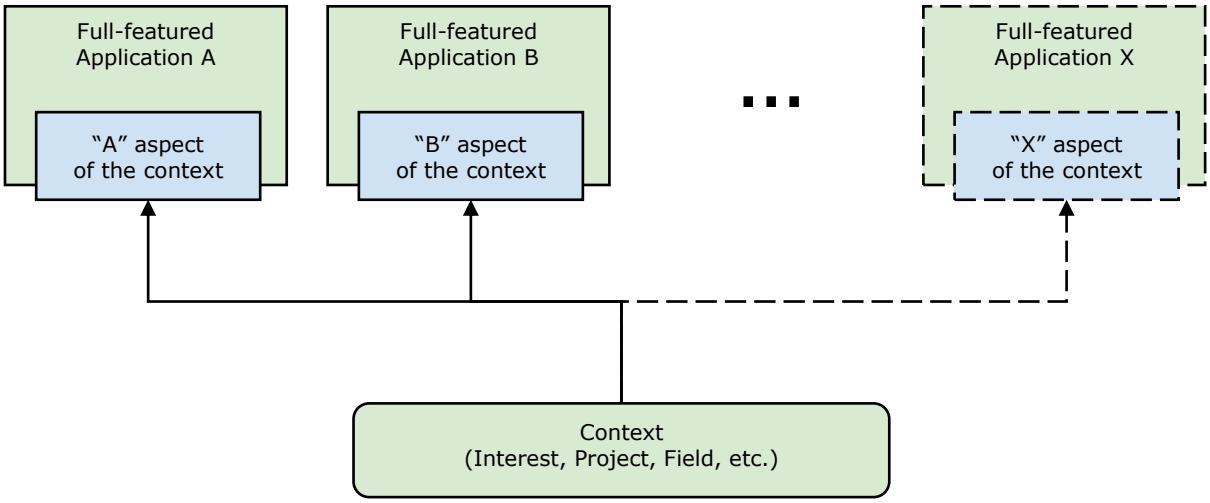
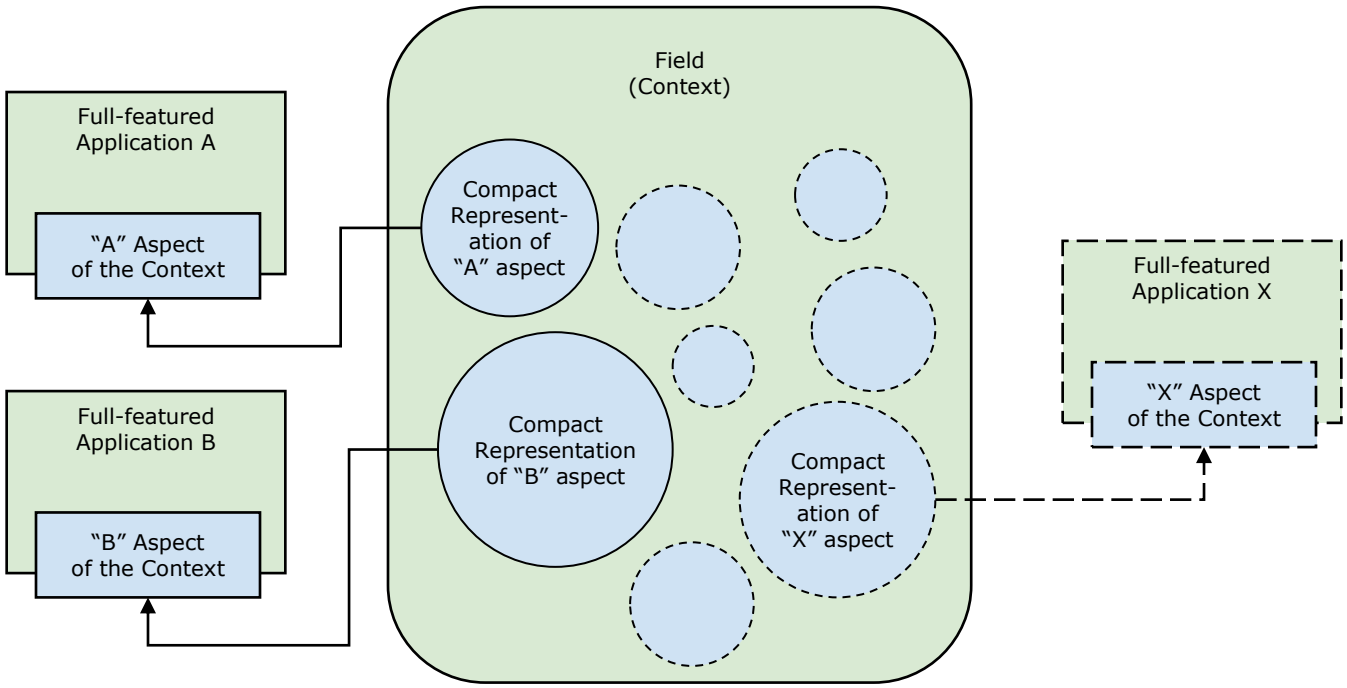
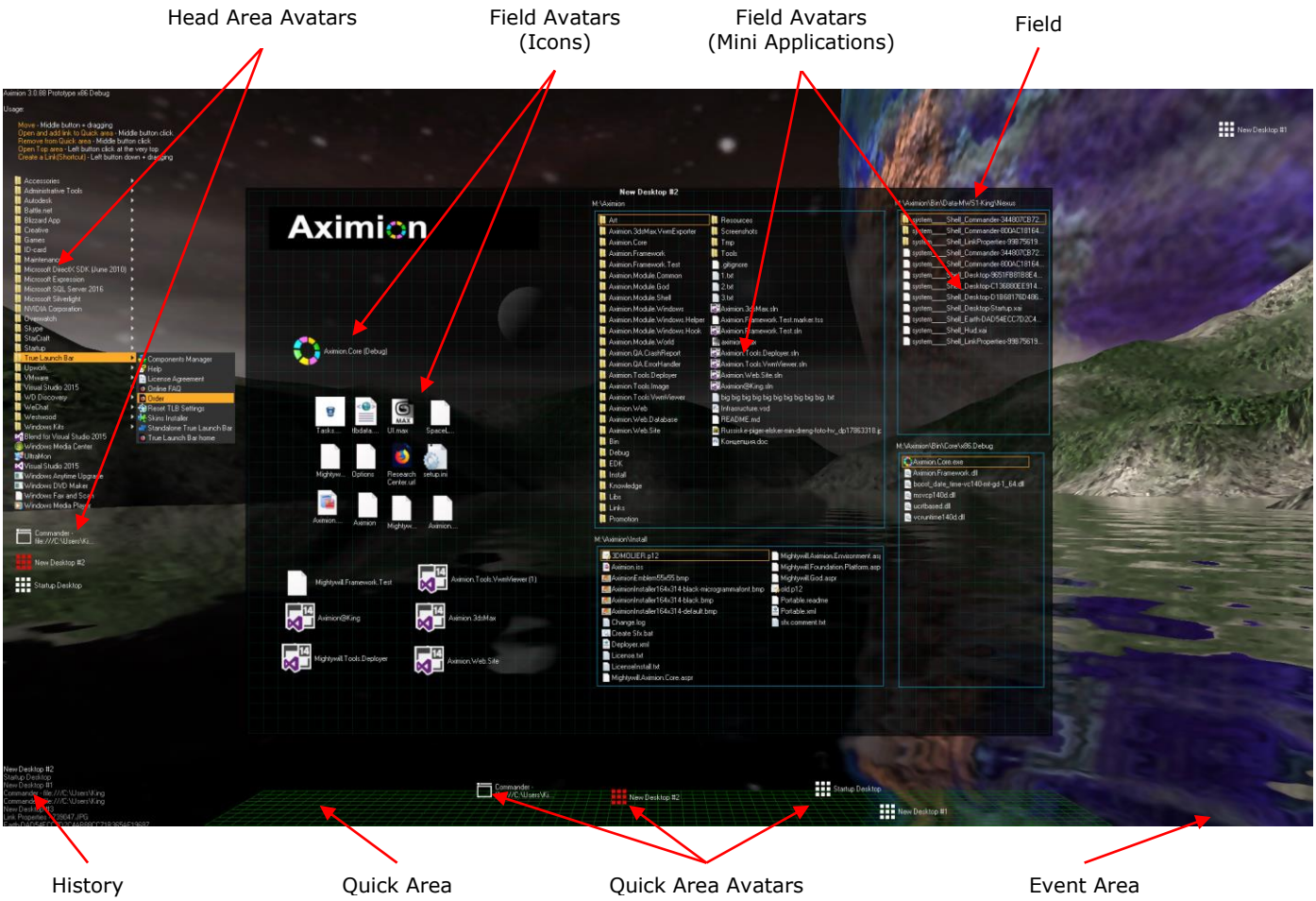


Figure 3. AMI Application-Context Paradigm



For example, if you have a particular project, you may want to gather all related files, links, and remote and other resources in one place and organize them according to your preferences. AMI fields look similar to regular desktops but can contain not only normal icons but also *avatars*. Each avatar represents some aspect of your particular context, project, or interest. They can change their look according to their importance to a user or for other requirements. The look of avatars may vary from simple icons to mini-applications with rich functionality more like mobile applications.

Figure 4. The Implementation of AMI



Any application UI can be attached to any other. For example, if a top-level UI element of application A is attached to a top-level UI element of application B, and then a user opens B, A shows up automatically. The user can then create various application unions that reflect some corresponding contexts. For example, if the user is a software developer then s/he can create an individual field for each of their projects and also create several other fields that would contain shared tools and resources required for all of those projects. Once any one of these project fields is open, all attached desktops with shared tools will appear together with it.

The Quick field is used to place links to objects required for the current session, in much the same way that tabs are used in browsers. A middle-button click helps the user to place an avatar in the form of a link to that area automatically. They do not disappear when the user closes the object, so the object is available for reopening when needed.

All avatars placed in the Head field are always available for the user and always stay on top of other objects. Using VR gear, this area acts as a heads-up display: i.e., it always stays in the field of view whenever a user's head moves.

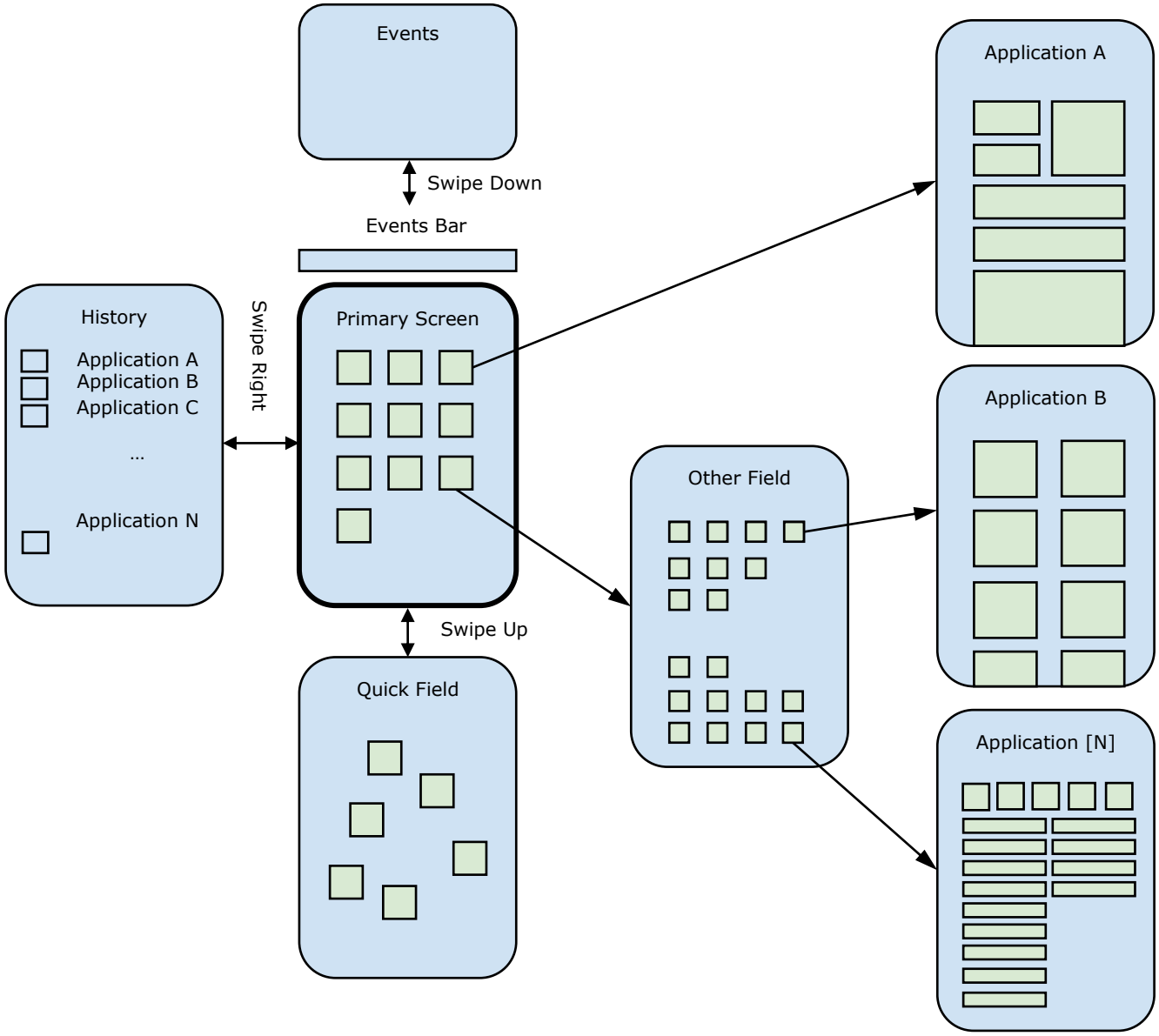
The History area is a history of user navigation. This function is similar to a browser history, and together with Quick Field effectively replaces the standard desktop taskbar behavior.

The Events area is for showing events and progress reports, similar to the desktop tray but with advanced functions. It can show simple notifications about events that require user attention. In addition, any user-initiated long-term operations should place representation items in this area so that a user has a continuous report about current time-consuming processes.

Mobile Mode

The technology is designed to support not only desktop and VR but also mobile environments. In this case, only compact representations (avatars) are used to display the application UI. The following diagram shows how the UI concept described above maps to mobile environments.

Figure 4. Mobile UI Mapping



Conclusion

UOS is designed to solve long-standing problems associated with application development by making a fusion of the best aspects of OS-native and Web UI paradigms. The technology allows the building of platform-independent applications that are as easily accessible as websites but as powerful as installable ones and with morphable UIs that are able to adapt to any existing devices as well as any future ones, thus on the one hand reducing developers' efforts to support different platforms and on the other hand giving users a new level of ease of use and friendliness in their devices and applications.

There is the potential to create applications that work on various currently incompatible Linux distributions, which would give a new lease of life to the open-source community.

This platform does not adhere to a particular OS, and once it is accepted worldwide, it will render a variety of existing operating systems unnecessary. Many years ago, platforms such as Windows and macOS brought us multi-tasking and graphical UI, but nothing revolutionary has been introduced since then. Besides cross-platform development, overlooked shortcomings of existing OSs also involve privacy, licensing, proprietary and patent issues. Thankfully, those days are now over. Initially launched as a platform-independent technology, UOS may finally require only one free, open, secure OS, thus ending the reign of big corporations in this sector of information technology.