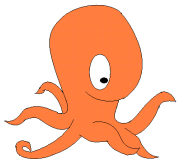

Building the Octopus

Francisco J. Ballesteros *Spyros Lalis* *Enrique Soriano*
12/21/2006
nemo@lsub.org lalis@inf.uth.gr esoriano@lsub.org

ABSTRACT

Octopus is the internal name for a system designed to provide ubiquitous access to computing resources. Its approach is unique in that the central idea to distribute the system is to centralize everything on a personal computer. Devices are later connected to this central system to provide distributed computing. This memorandum describes the initial design and underlying ideas for the new system.

1. Problem Statement

Today computing environments are complex. They are made of a miriad of devices and machines interconnected through multiple networking technologies. They are heterogeneous, dynamic, must support network partitions, have administration problems, run different sets of system software, have different applications and capabilities, and the list goes on. Many of these problems are increased because we are considering a *distributed* system made of *heterogeneous* and *volatile* components.

However, on a globaly connected world, the environment is no longer distributed, is no longer heterogeneous, and is no longer volatile.

Google [4] is an excelent example that can be used to support this claim, i.e., to justify that we would rather be no longer pursuing distributed, heterogeneous, and dynamic environments. It centralizes the implementation, and provides ubiquitous access to it. Users open their browsers, address them to *pages.google.com* or to *calendar.google.com* and start working. We argue that we can apply the same idea to the whole operating system.

The google file system [4] is another example. It centralizes control, to handle with a much more simple, centralized, implementation the miriad of distributed storage and computing devices used to implement the service. It builds a distributed file system by centralizing its core algorithms and control, and distributing data. The same underlying idea could be applied to the system software. We could stick to a single system, centralizing control, but alowing the system to span all the distributed devices of interest.

Systems like WebOS [9] tried to do this. Arguably, it seems they failed. But we argue that we just have to try harder. Our proposal is to adopt the Internet as the system bus, to plug any device in the system.

2. Computing Environment

The computing environment considered for the target system is depicted in Figure 1. It reflects reality as of 2006, and is expected to be even more common in the upcoming years.

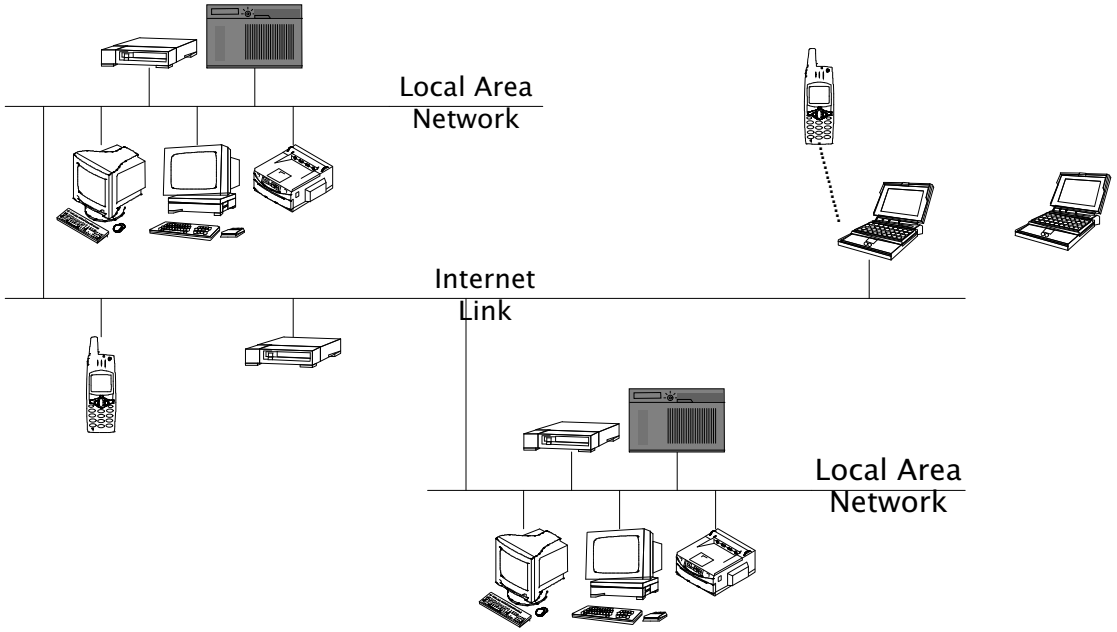

Figure 1: *Computing Environment for the new system*

The first assumption is that **all computers are connected** through some kind of internet link. This includes not only computers, but also any computing device (e.g., DVD players) that provides one or more resources (e.g., playing DVDs).. The resources are arranged so that there are islands connected through high speed local area links (e.g., Gigabit Ethernet, FastEthernet, or fast 802.11 WaveLan networks).

Resources within a single LAN are very well connected and have no latency or bandwith problem at all. However, for a given user and time, there are multiple islands of interest. The links between different LANs are expected to have poor latency and bandwidth, although they are expected to be roughly equivalent to today DSL lines.

In general, most resources are located within a LAN, colocated with many other ones. However, it is also feasible that some devices of interest for the user are either isolated, connected directly through WAN, DSL, Bluetooth, or other poor links.

We consider only a single exception to what we have said. In some cases, users may want to use only a single device, disconnected from the rest of the world.

The second assumption is that **there is plenty of computing capacity** on most computers. This means that the power of a single computer suffices for most if not all the tasks of interest for the user. Note that we refer to full computers, and not to handheld or specific purpose devices.

Our last assumption is made regarding the user needs. We assume that the user wants to **use his/her devices with his computer.** In general, provided that the user is connected to his/her personal computer, the user wants simply to be able to use the

software (both system and applications) while being able to employ whatever device may be in the network.

3. System description

There is a **single dedicated computer per user**. We refer to it as *the computer*. If a user has more than one (which is the common case), one of them is designated as the computer. All user programs execute on the computer, irrespectively of the user's location and of the devices and resources required to run them.

In this system model, the computer does not have any input/output resources of its own; we think of it as a box with lots of (virtual) memory and processing power. If there are I/O devices, they are considered to be attached to the network and not to the computer itself. The scheme is depicted in figure 2.

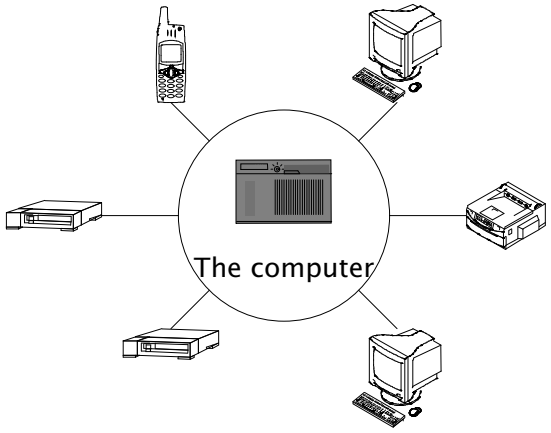

Figure 2: *System Model: The network is the bus to connect devices to the single computer*

The computer is considered to be highly available. This can be achieved either by installing watchdogs to restart it after software failures and/or by replicating some of its components (e.g., by implementing it using two or three separate computers that are considered as a single one from the rest of the world).

Programs running on the computer employ resources, such as storage, bitmap screens, keyboards, pointers, audio, image and video recorders and players. These resources are provided by peripheral devices or by applications running on other systems (in this case they are considered as devices and not as applications by our system).

We make a difference between devices and resources. A *device* is a piece of hardware, perhaps part of a computer, that provides one or more services. In some cases, an application program of a traditional operating system may be also considered as a device. For example, a DVD reader, an audio card, a MP3 player program, etc. are devices for the octopus. A *resource* is an artifact that provides a certain service. For example, to play MP3 files, a player resource is needed. The difference between device and resource is clean if we consider that a single device (e.g., an audio card) can provide more than one resource (e.g., speech, MP3 player, and text recognition).

The devices providing such resources can be stationary or mobile, and communicate with the computer via a file system protocol [6]. Although devices can be highly heterogeneous, are distributed, mobile, and can be switched on and off at any time, the computer is a single, central, homogeneous, system where all the system software runs.

When resources are provided by other computers using other operating systems, we export just the resources of interest indenpendently of the native system of the foreign computer. In this sense, this alien system is just more hardware for the octopus.

Access to resources is transparent with respect to the “physical connection technology” used to attach them to the computer. This is similar to using USB over different link technologies. In our case, octopus can use the 9P file system protocol (plus some extensions) over IP or other transports.

In the computer, programs see no difference between accessing a resource via a local hardware bus, a system extension bus, external serial and parallel links, and TCP/IP. type of network, including the Internet. For us, **Internet is the system bus**.

This means that the interfaces between resources and the rest of the octupus must be of a high-level of abstraction and must be designed with poor links in mind. We map these interfaces into virtual file trees, following the Plan B aproach [3], which is indeed a follow up of the ideas in Plan 9 [6] and UNIX.

4. Benefits and Drawbacks

The proposed system model has several benefits. The most important one is *simplicity*. Because all the implementation is kept centralized, the system and the applications can be kept simple. There is no need for distributed garbage collection, coordination amount heterogeneous and distributed systems, complex peer to peer protocols, and so on.

Another important benefit is *ubiquitous access* to the system, because most popular systems are to be considered resource providers for the octopus. This includes web navigators as user interface devices. Unlike in other descendants of WebOS, the octopus permits using local devices as well (not just the user interface within the web page).

There will be *fast boot* and suspend/resume, because the system will never shutdown. Only, external devices may be disconnected and reconnected. But both the system and applications will continue operation despite this fact.

The system is easier to protect, provided that devices are secured. This follows because there is only one system to protect. In the end, system security depends on which devices (linked to the octupus) are able to obtain or supply data for the system.

Administration gets simplified. There is only one system to maintain. Everything else are stateless devices. In fact, the central computer might be provided by a third party responsible for its adminstration.

There are also some drawbacks. The most important one is that it will be easy to *waste resources* because that is deeply assumed by our design. A countermeasure is to try to exploit idle resources by making them available to the computer. Of course, most of the code to run user interfaces would be running at peripherical devices and computers, relieving the central computer from that task. Nevertheless, we strictly follow our principle that *all* the software runs in the central computer, including all applications.

A failure in the central computer renders the whole computing system useless. This means that measures must be taken to make it highly available. Because software failures are more usual than hardware faults, watchdogs used to restart the system upon failures might suffice.

Because resources are assumed to be connected to the computer, this may lead to expenses due to connections used by mobile devices. In some cases, the mobile device might work isolated, but our approach tends to keep it connected at all times with the central computer.

Device interfaces must be of a high level of abstraction, because otherwise they would make the system unbearingly slow when the bus is indeed a WAN connection. This means that there can be problems to access low-level interfaces.

5. System Resources

This section introduces some important resources for the operation of the system, and describes the main design guidelines for all of them. In general, all resources are exported using small (virtual) file trees. These file trees are imported into the computer using a import mechanism similar to that in Plan B.

5.1. Copy

Our model is to let resources (on devices) to be accessed only from within programs that execute on the computer. In theory this is enough to do anything. In practice, it is important to support direct device-to-device data transfers between devices, to relieve both computers and the network from a lot of useless work and traffic. In some cases, this may even be the difference between being able to perform the requested task or not being able to do that.

The typical example scenario is to copy a (perhaps large) file located on a DVD reader to a nearby device, e.g., a DVD player. Clearly, it is desirable to do this with a minimal involvement of the computer. Note that the computer might be poorly connected to both devices, although both devices might be well interconnected as shown in figure 3.

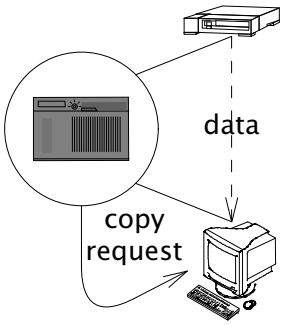

Figure 3: *The computer asks for a copy, but bytes can be transferred between devices.*

A lot of flexibility can be achieved by introducing a *copy resource* loated at the target device. This resource provides a service to read bytes from another resource and then (locally) copy them to a local resource. This means that most devices must feature a local program for performing this task. Adhering to our resource-based model, we camouflage this in the form of a special copy resource. You can think of the copy device as a network-enabled DMA device. The copy device permits transferring parts of resources (i.e., files), to support streaming and the user request blocks while the requested transfer is in progress.

The operation of this artifact is best shown with an example. Figure 4 shows two resources and a copy resource that can be used to transfer data between them. As we said before, all resources are provided through file based interfaces.

In the figure, there are two devices involved (depiced by rectangles). Colocated with the target resource is a copy resource. The source resource is at a different machine. Assuming that a name space in the computer has the target device mounted at `/target` and the source device at `/source`, we can establish a copy as follows:

```
(1)     ; taddr=‘{cat /target/addr}
(2)     ; saddr=‘{cat /source/addr}
(3)     ; copyop=‘{cat /dev/random}
(4)     ; echo $taddr 0 $saddr 0 0 > /copy/$copyop
```

The example shell session reads (line 1) `/target/addr` to learn the address of the

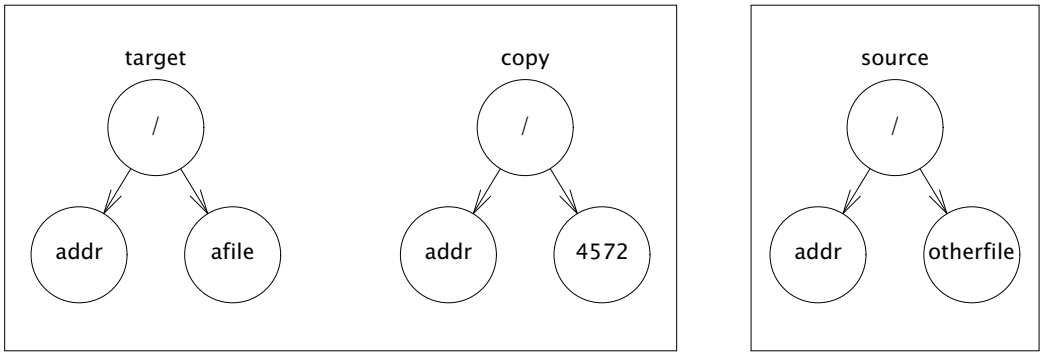

Figure 4: *Files involved in a copy operation between two devices.*

target resource. An address is a network address plus an optional path and file tree spec (similar to volume addresses in Plan B). The same is done for the source (line 2). All resources must have a `addr` file to support copy.

Creating a file in the copy resource (which is assumed to be already mounted) and writting into it the destination address, the offset into the resource file, the source adress, the offset for it, and the maximum number of bytes to transfer (0 for all of them), initiates a copy between the devices as shown in figure 3. The write operation blocks while the copy is in progress. File names are randomized (line 3) to avoid name clashes.

While the copy is in progress, the copy device can provide a status file (e.g., `4572.status` in the example) that reports the last offset copied on each read.

The copy resource must be registered with the computer and requires a capability-like protection scheme. We assume that there is a single copy file system mounted in the computer, that interfaces to remote copy resources. This makes things more simple for the rest of the system.

As a convenience, a library function with the prototype shown here

```
ulong copy(char* to, vlong tooff, char* from, vlong fromoff, ulong count);
```

can be provided to give access to the copy device in a simple way. This function can determine the resource addresses and use the copy device without placing the burden on the user. Besides, when the devices do not support the copy operation (i.e., do not provide the `/addr` file on their trees), the implementation can fall back to using `read`(2) and `write`(2) and the application would not notice (but for the lower speed).

Note that the computer may now request one resource to retrieve data from another without the intervention of the computer. The access control scheme introduced below tolerates this behaviour without requiring additional mechanisms or protocols.

5.2. Storage

A storage resource has the form of a (remote) file system like in Plan 9 and Plan B. Any device with storage capacity can provide a storage resource. Pluggable media in foreign systems, such as disks, USB sticks, and CD/DVD readers, can also be turned into (proper) storage resources, via wrappers that adhere to the (file-based) resource registration and access protocol and (dynamically) register themselves with the computer.

Because storage plays a key role in personal computing, a basic and reliable storage resource must be always connected to the computer, acting as its main program and data repository. In most cases, the computer would page binaries only from this

resource. The storage capacity of the repository can be increased in a straightforward way, simply by adding more (reliable and always connected) storage resources (e.g. disks). An appropriate backup/replication mechanism is required to tolerate failures.

Of course, any portable device can contribute with less reliable and/or not always available storage resources. Since these resources may "appear" and "disappear" dynamically, they are more likely to be employed by people in causal recreational and entertainment activities, or to hold mostly data and not programs.

It is also important to note that the volatility of such storage resources is non-critical for program execution. For this reason the system does not have to provide any advanced storage resource management. Programs interested in exploiting volatile storage resources must explicitly discover them at runtime and must also be able to tolerate failures when trying to access them. If a program (or the user) wishes to guarantee that a particular storage resource (or file) will be "always accessible" then it must move/copy it to the computer's repository. Ideas from both Venti [7] and Omnistore [5] can be applied here.

5.3. Caching

Provided that it would be common to have the setup shown in figure 3, where data transfers may be performed between devices, it is convenient to cache data at the target of the copy operation. This can speed up system operation when the connection between the target and source devices is not within a single well-connected island.

In general, caching can be performed by the copy device itself, by keeping a persistent cache on a near-by storage resource. This takes advantage of resources outside the central computer that are always wasted by our design.

When a copy operation is sent to the copy device, the device issues a *stat* to the source resource. If this resource is up-to-date with respect to the cached copy, the copy proceeds within the machine where the copy device is located. Otherwise, the resource is read and the cache updated. To maintain coherency, the cache is always write-through and in all cases checks out that the source is indeed the cached data.

An interesting device to place caching into is a USB memory stick. This can be considered as a *network enhancer* for the octopus. The user may roam and, when using a remote terminal built out of devices far away from the system, plung its network enhancer in the terminal. The copy device would locate its cache and use it.

5.4. User Interfaces

User interface resources are employed to allow an application program (running on the computer) to interact with a user via (portable) devices. The computer does not have any reliable and always connected device to support the UI. All I/O resources used for UIs are provided by external devices that are "attached" to the computer in an ad-hoc fashion. As a consequence, the type and amount of resources (e.g., screen sizes and number of screens) that are available to the computer vary in time. It is the job of the system to assign the available UI resources to the currently running applications.

The abstraction of UI resources is done along the lines of omero [2] in order to maximize flexibility and versatility while minimizing communication requirements between devices and the computer. This is so to let our model work over long distances (bad latency) and for a large number of concurrently running applications on the computer (poor bandwidth).

5.5. Sensors and Actuators

Devices intented to sense the physical environment and to actuate on it are to be handled like other resources, by modelling them as virtual file trees and then plugging them into the (single) computer. For an example of how Plan B did this, refer to [1], but note that now there is only a single computer. All user automations for the environment would simply run on it.

5.6. Other resources

In general, we require either one or more abstract interfaces for each resource. This is a consequence of trying to minimize latency and bandwidth needs. For instance, for audio, we would export serveral diferent resources for the very same audio output device:

1 A voice output device accepts text to be spoken. This is very efficient for talking to the device across the network. A voice synthesis program runs locally near the output device and therefore we both relieve the computer from performing the synthesis and the network from transfering all the raw audio.

2 A MP3 output device accepts MPEG 2 Layer 3 encoded files. This is better than transfering raw PCM samples.

We omit the discussion of other devices, all of them must be reworked along the same lines.

6. Security

The user must be able to use his/her computer (programs) in conjunction with his devices (resources). Some times, the user would borrow or lend resources on devices of other people. Also, it must be possible to support the notion of public-domain devices which could provide resources to anyone wishing to use them.

We are considering using capabilities to meet these requirements. The scheme would be similar to other capability-based file system security architectures.

6.1. Access Control

All devices owned by a user are given appropriate keys and credentials that allow them to perform the following tasks:

1 Prove to the user's computer that they are owned by the same user

2 Encrypt data exchange with the computer;

3 Check that a capability is appropriate for the requested resource access.

4 Prove to other devices that they are owned by the same user.

5 Encrypt data exchange that takes place directly between devices.

6 Authenticate the computer. Authentication has to be bi-directional.

For each resource provided by a device that belongs to the user, added to the computer registry, the system generates a special *passe par tous* capability. This can be later handed over to any program that wishes to access the resource. The program then sends the capability along with access request to the resource, which performs the requested operation if the capability is appropriate.

6.2. Delegation

Say user A wishes to allow user B to access a resource R on device D (owned by A). This can be achieved as follows: (1) A asks B to give him the name of his computer; (2) A uses a special resource access control application (running on A's computer) to generate a capability for accessing R on D for any program that runs on B's computer; (3) A's computer generates an appropriate capability; (4) A's computer informs device D

(resource R) about the existence of this new capability; (5) A's computer sends a resource registration for R on D along with the capability, using the standard resource registration protocol, to B's computer; (6) B's computer creates a new resource registration for this particular resource and associates it with the capability received; (7) resource R on device D of A becomes visible to B's programs .

Capabilities generated for programs running on the computers of other people may be given an expiration date when creating them. However, timeouts are not enough if A wants to (suddenly) stop B from accessing R, e.g. as soon as B walks away from A. In this case A must explicitly revoke B's capability. Given the above approach, this can be achieved as follows: (1) A uses the resource access control application to cancel B's capability for R on D; (2) A's computer informs device D (resource R), which aborts any ongoing access requests that have been made using this capability (and refuses to accept any new access requests that carry this capability).

This approach can be nicely extended to accommodate public-domain devices, as follows. Each public-domain device features a special locally running resource control program that is waiting to accept the name of a user's computer. As soon as this is done the program issues appropriate capabilities for its local resources, and registers them with the user's computer (as above). These capabilities are automatically revoked (as above) upon timeout or as soon as the name of another computer is input.

6.3. Proposed Scheme: Copy Operation

The security scheme can be based on SHAD [8] and capabilities that are used in order to permit the copy operation among devices. Capabilities are nothing but keys. These keys are generated, used and discarded.

The copy operation is performed this way:

1 The computer sends a key K plus a (very short) number P to the the target device writing the data in a file named cap served by its FS.

2 The computer sends the same key K to the source device, plus a file name N and the short number P. It writes the data in the file cap of the source FS.

3 Let Port be the port in which source FS is listening. The target device opens a 9p connection to Port+p, and it starts the 9p protocol ciphered with the key K. This is a 9p connection without authentication.

4 The source FS can decrypt the connection, because it knows K.

5 The source FS knows the name of the requested file, N. Therefore no other file can be accessed. The FS ignores operations such as walks etc. Attach's FID is ignored, and the attach is performed over the FID representing the root of n. When an open is performed, it doesn't care about the specified FID. It opens the file N. Then, the only allowed operations are sequential reads until EOF and clunk.

6 Target device reads and clunks the file.

7 Source FS includes K in a black list. K cannot be used anymore (within a reasonable time) to access this FS.

P allows to have concurrent copy operations in a single device. K adds security (authentication+confidentiality) to the operation. Communication between the computer and the devices is secured with a previous set device secret, the same way that SHAD's terminal key.

6.4. Working with one device that does not belong to the user

Let TCa be the computer of user A, and TCb the computer of user B. Db is a device that belongs to B, and Da belongs to A. TCa and TCb share a secret, a pairing key. When A needs to work with Db as target and Da as source:

1 Db sends to TCa a confirmation message to start the protocol. Confirmation can be avoided through configuration, but it's needed by default. This message must be authenticated, and we can use a protocol like the one described in SHAD (terminal sharing protocol).

2 If A confirms the operation, TCa asks TCb for permission and sends P and N. User B can cancel the operation in this very moment if he wants to. Confirmation can be avoided through configuration, but it's needed by default.

3 Control access to devices can be performed by SHAD's roles, that is an ACL and RBAC mixture.

4 If B grants access, TCb generates K and writes it in the file cap of Db's FS, together with P and N.

5 TCb sends K to TCa.

6 TCa writes K to Da's cap file, together with P and N..

7 Db opens a 9p connection with Da, just like the previous case (described above). It reads and clunks the FID. K is discarded.

The same protocol can be performed when Db is the source and Da is the target.

6.5. Confirmations

Users have a mobile device that always carry on, a-la UbiTerm. This device has a key assigned and has a open (encrypted) connection with TC. The device runs Oshad, a graphical front-end for SHAD. Confirmations are performed in this device. It also emits notifications to the user.

6.6. Enhanced capabilities

We could include additional restrictions to the capabilities. A counter C can be used to permit the capability be reused many times. We can also use a timeout T to make the capability expire. These variables could be sent together with K, P, and N.

Note that these restrictions has to be generated by the computer of the owner of the source device, and they must be sent directly from the computer to the source device's FS. If not, the could be faked.

6.7. Physical Security

Capabilities are (safely) kept on the computer, not on (portable) devices that can get lost or stolen. However, this does not make the system secure. More specifically, if person A manages to physically access any device of person B, he/she can use it to launch and interact with any application on B's computer, i.e. gains access to the entire system of B including all resources owned by B; as well as resources of person C to which access was (temporarily) granted to B.

This is a problem, but it seems unavoidable and there is a possible solution: The user may wear a small device that periodically sends out encrypted beacons with his/her id and a sequence number. The devices owned by the user would listen for such consecutive beacons and shutdown as soon as they stop receiving them. For more flexibility, the user can configure some devices to remain operational even if they do not receive these beacons; for instance, devices that are located somewhere safe (home, office, etc).

Note that this idea forces the user to wear a device at all times (as long as he/she wishes to use the computer). And then, this device may be lost or stolen too; or just break. Therefore, this is not a real fix, but a workaround.

There is another, more or less orthogonal, solution. Programs which are used via certain portable devices are allowed to access only a few non-critical resources. A

program receives full access only if it is used via distinguished devices, assumed to be located in a safe environment. One could also use any device to run a special authentication program that, if successful, issues privileged capabilities that temporarily allow programs used via this device to access protected system resources. Notably, this means that access control in made context-dependent. It would depend on which devices are being used to interact with, and provide input to, this program. This requires refining the access control scheme introduced previously.

7. Problems and Scenarios

7.1. Application test bed: Player

To test the system ideas and be honest to the real world, we focus on how to provide services for a particular and concrete application, a music player.

The player program runs at the computer, and uses available devices to perform its job. In particular, songs can be selected from storage volumes for music files. The system should be responsible for locating such volumes and making them available to the player. A set of controls permits the user to drive the program. These controls can be replicated and/or moved around different UI services, like in omero.

Besides, the player must be able to use available screen space to display the poster of the CD, or any image related to the music being played. The same can be applied to the lirics.

Regarding audio output, the player must use an appropriate audio device to stream the data for its reproduction. It is important to note that the connection between the music storage volume and the audio output device may differ in quality from that between these devices and the computer. Therefore, bandwidth and latency must the considered precious.

Once started, the user must be able to quickly locate and display the player interface, perhaps at a different screen. Also, the audio output device might change during the execution time for the player.

Music volumes may come and go. Therefore, the selection of available music is subject to change as well. And the same applies to any other resources used by the player. The only constant is the single computer used to run all the software.

7.2. One interface does not fit all.

The user may employ widely different UI devices to interact with the application. When started for the first time, the application passes to the central UI service running at the computer the whole, full-fledged, UI structure. Afterwards, the user can emply the system interface to trim, adjust, copy, and/or replicate the application's UI structure to fit different (expected) usage scenarios that involve a variety of devices.

7.3. Dynamic User Interfaces.

The player application may need different interfaces depending on its execution stage (e.g., to select songs, to select output, and to control the player). Properly staging the user interface of an application (rather than trying to instantiate it "all at once") is particularly important when there are few UI resources available. This can be achieved by letting an application explicitly issue different UI structures to the system, depending on its (internal) state, at runtime.

It is important to keep in mind that one very important state is when the application does not require any UI resources at all, and can happily work in the background.

7.4. Distributing UI resources.

What makes things complex is that we may have many concurrently running applications and many different devices providing many different UI resources, and that some UI resources may be able to be used concurrently by different applications (e.g. big display that can be subdivided into several separate sub-displays, which is practically what happens on a desktop environment).

Note that we have a problem for both the cases of resource scarcity (which applications should be given priority over others?) as well as resource abundance (which are the best UI resources for each application?). Both problems require the system to provide a mechanism that would let users employ different policies, perhaps by scripting.

References

1. F. J. Ballesteros, G. G. Muzquiz, E. Soriano and K. L. Algara, Traditional Systems can Work Well for Pervasive Applications. A Case Study: Plan 9 from Bell Labs Becomes Ubiquitous., *Percom*, 2005.
2. F. J. Ballesteros, G. Guardiola, K. L. Algara and E. S. Salvador, Omero: Ubiquitous User Interfaces in the Plan B Operating System, *Proceedings of IEEE PerCom. Also at http://lsub.org.*, 2006.
3. F. J. Ballesteros, E. S. Salvador, K. L. Algara and G. Guardiola, Plan B: An Operating System for Ubiquitous Computing Environments, *Proceedings of IEEE PerCom. Also at http://lsub.org.*, 2006.
4. S. Ghemawat, H. Gobioff and S. Leung, The Google File System, *19th ACM Symposium on Operating Systems Principles*, 2003.
5. A. Karypidis and S. Lalis, OmniStore: A System for Ubiquitous Personal Storage Management, *Proceedings of the 4th IEEE PerCom.*, 2006.
6. R. Pike, D. Presotto, K. Thompson and H. Trickey, Plan 9 from Bell Labs, *EUUG Newsletter 10*, 3 (Autumn 1990), 2-11.
7. S. Quinlan and S. Dorward, Venti: a new approach to archival storage, in *First USENIX conference on File and Storage Technologies*, Monterey,CA, .
8. E. S. Salvador, F. J. Ballesteros, K. L. Algara and G. Guardiola, A Human Centered Security Protocol for Ubiquitous Environments, *Submitted for publication. Also at http://lsub.org.*, 2004.
9. A. Vahdat, T. Anderson, M. Dahlin, D. Culler, E. Belani, P. Eastham and C. Yoshikawa, WebOS: Operating System Services For Wide Area Applications, *Proceedings of the 7th HPDC.*, 1998.