

# Software Architecture

## Review Plan 9

### **Authors:**

Theodoros Polychniatis

Sander van der Rijst

Ruben van Vliet

Geert Wirken

January 2011

Utrecht University

Faculty of Science

Department of Information and Computer Science



**Universiteit Utrecht**

## Table of Contents

1	Introduction .....	3
2	Change impact analysis.....	4
2.1	Changes with small impact .....	4
2.1.1	New types of hardware.....	4
2.2	Changes with big impact.....	4
2.2.1	Adding new types of hardware to Plan 9.....	4
2.2.2	Performance.....	7
2.2.3	Security .....	7
3	Evaluation .....	9
3.1	Strong points.....	9
3.2	Suggestions for improvements .....	9
4	Conclusion.....	10
5	References .....	11

## **1 Introduction**

Plan 9 is a distributed operating system which began in the late 1980s as an attempt to have it both ways: to build a system that was centrally administered and cost-effective using cheap modern microcomputers as its computing elements. The philosophy of plan 9 is: “everything is a file”; a concept also known to Unix. Furthermore, another key aspect of plan 9 is that there should be no difference between local and remote objects for the users.

Plan 9 was developed primarily for research purposes as the successor to Unix by the Computing Sciences Research Center at Bell Labs. They considered the problems with UNIX were too deep to fix, but some of its ideas could be brought along. A network-level protocol, called 9P, was designed to enable machines to access files on remote systems.

Furthermore, a naming system was built which lets people and their computing agents build customized views of the resources in the network. This is where Plan 9 diverged from other similar like systems: a Plan 9 user builds a private computing environment and recreates it wherever desired, rather than doing all computing on a private machine.

The design is built around three principles. First, resources are named and accessed like files in a hierarchical file system. Second, there is a standard protocol, called 9P, for accessing these resources. Third, the disjoint hierarchies provided by different services are joined together into a single private hierarchical file name space.

A handful of companies have had success in selling Plan 9-based products. Most notable is Vita Nuova, which continues to maintain and market Inferno, a Plan 9 derivative targeted at set-top boxes and other embedded devices.

The designers of Plan 9 claim that the system is easy to use for programmers, and is an excellent example of high quality system design and software development. Studying its code reveals how simplicity can be more effective than contortions made by other systems.

This report will review the architectural description of Plan 9, written by Team F during the Software Architecture course of University Utrecht. It discusses the architecture from the lowest building blocks based on the provided requirements and quality aspects. This evaluation has been conducted by Team E and can be considered as a peer review.

A change impact analysis is used to conduct the review. It attempts to expose the impact of a given change on some artifact X to another artifact Y. The focus lies on the requirements, e.g. which architecture descriptions are being influenced when requirements change.

The goal is to describe requirements which cause small changes in the architecture and to describe which new requirements would cause the architecture of Plan 9 to fall apart. Finally, we will assess if these questions can be answered by the architecture description of Team F. The gaps will be pointed out in chapter 3.

## 2 Change impact analysis

A change impact analysis attempts to find out the impact of a change on some artefact X to another artefact Y of a system. As a case study, we try to estimate the consequences of design changes in Plan 9.

Bohner and Arnold defined a change impact analysis as follows: "*identifying the potential consequences of a change, or estimating what needs to be modified to accomplish a change*"

A three steps approach has been followed to identify potential changes:

- From the paper, we identified artefacts which could be modified.
- We tried to trace relationships which would indicate a dependency on the artefact.
- If a change affects the dependent artefacts, then other artefacts that depend on these dependent artefacts will also be affected.

After these three steps, we will investigate the implications of the impact analysis improvement.

In this section we will present the identified artefacts and its implications of change. We will describe several requirements changes which would cause the architecture of Plan 9 to fall apart. That is, we try to find what the system should sensibly be able to do, but only can be accomplished with much additional re-architecting.

### 2.1 Changes with small impact

#### 2.1.1 New types of hardware

Adding a new type of hardware to Plan 9 is possible with the current architecture of Plan 9. All hardware interfaces are located in the `/dev` subsystem, and for a new type of hardware, one could just add a new file as an interface to the hardware.

For instance, say that joysticks are not supported in Plan 9 and a developer wants to develop a driver for the joystick. He finds that there is no file `/dev/joystick` yet and decides to use that file as an interface to the joystick, by creating a new file server that acts as a mediator between the `/dev/joystick` file and the joystick itself (possibly by keeping track of button states etc.).

However, there does not seem to be a specific naming standard on the files in `/dev`. Filenames should of course be unique, but there are no strict guidelines on the naming of such files.

### 2.2 Changes with big impact

#### 2.2.1 Adding new types of hardware to Plan 9

In Plan 9, all communication between hardware and applications is done via files. This is a type of abstraction that can simplify development and debugging, but it can also be problematic when new types of hardware arise or when existing types of hardware (e.g. computer mice) have new capabilities.

The communication between hardware and application is done through files, using a (human readable) text protocol. For instance, the position of the mouse is stored in `/dev/mouse` and an application constantly reads out `/dev/mouse` to determine the position of the mouse cursor. The position of the mouse is returned in the format that is specified at some time, namely an array containing the (x, y) position of the cursor and the state of the buttons (pressed or not pressed).

It is important to note that there is no formal protocol version specified when opening the `/dev/mouse` file. This possesses a problem: it is difficult to add new information to the mouse interface when new elements are added to a hardware device. For instance: a computer mouse may be equipped with a scroll wheel, which was not foreseen in the original specification for the mouse driver.

When the interface between applications and mouse was done using a library, the developers of the mouse driver could just add a new function to enable application developers retrieve information about the mouse (e.g. the position of the scroll wheel). However, the approach that is taken by the Plan 9 developers makes it more difficult to implement such changes, as the protocol of `/dev/mouse` should be changed (extended) and all applications that use this protocol. Of course, there are libraries available that provide an abstract interface to the mouse (which might accommodate the need to change many applications), but this goes a bit against the idea of Plan 9 ('everything is a file'). In Plan 9, directly communicating with files is advocated over using libraries.

Changes in protocols can probably be mitigated by offering different versions of a protocol (e.g., a `/dev/mouse` and a `/dev/mouse2` in which the latter one offers support for scroll wheels), but this isn't a very clean approach as the whole `/dev/` file system would be cluttered with old version interfaces. On the other hand, one could mitigate the problem by effectively rewriting all applications that use a certain interface. This would probably not be feasible. A third option is to make the protocol that is employed in an interface file easily extendible. This is not really the case in the `/dev/mouse` example. A typical read of `/dev/mouse` is shown below:

```
m 670 66 0 2257710 m 676 68 0 2257730 m 677 74 0 2257750
```

As can be seen in the example above, an update of the mouse information is preceded with the character 'm' followed by an ordered list of data (divided by spaces). The data that is provided is (sequentially) the coordinate of x and y, the state of the mouse buttons and a timestamp.

Using this protocol, it is difficult to add new features in a flexible manner. The order of the data is important, meaning one cannot insert a new element before another element. This also implies that abandoned features cannot be removed from the protocol, they have to reside in new versions of the protocol to preserve the existing order.

Another (arguably better) solution would be to add the names of a data element to the list of data elements, or to store this information somewhere in `/dev/`. This means that developers can adapt their applications and use a flexible protocol: only information that is really available is provided, and new

information elements can be added relatively easy. And even more important: this preserves backwards compatibility with older applications.

An example would be something like:

```
m x:670 y:66 b:0 t:2257710 m x:676 y:68 b :0 t:2257730 m x:677 y:74  
b:0 t:2257750
```

And for mice with scroll wheels, the driver would return information in the following format :

```
m x:670 y:66 b:0 t:2257710 sy:3 m x:676 y:68 b :0 t:2257730 sy:4 m  
x:677 y:74 b:0 t:2257750 sy:5
```

When in the future the scroll wheel is abandoned over another system, the format can easily be adapted to remove the descriptor ('sy') of the scroll wheel position. New features can be added relatively easy (for example support for a sixth mouse button).

### **2.2.2 Performance**

The developers of Plan 9 very wisely designed and implemented an architecture that can bind many remote computers/devices so as to operate as one. Their target group is not the simple desktop user but the one who has many requirements in terms of computing resources. This is the main reason that a user would not install an operation system in one, but in several machines. One of the most demanding categories of applications that require such operating environments is the image rendering applications category. The video processing software makes heavy use of most resources of a system like storage, CPU, network, graphic components etc. Another type of demanding software that uses similar amount of resources is the simulation applications. Both these software types need platforms that provide quality aspects like reliability and performance.

High level of performance is a very hard goal that every operating system has to achieve. Plan 9 is supposed to be efficient, but it has not been proved with demanding applications. The fact that the architecture resembles FUSE, which “works well” as the authors of the architecture paper write, does not guarantee that it is efficient. Moreover FUSE is not so popular for its performance; it is rather popular for the modularity that has provided to Linux systems to abstract the mounting of different storage systems. Plan 9 has the disadvantage in this case that it needs many function calls between different computers for simple operations. The authors also write that there is not much overhead in these calls but still they are not as fast as the static function calls.

Therefore the question is that if the developers of Plan 9 want to satisfy the demanding users of simulation and video processing software how the architecture can address the performance issues that will arise. It is a challenge to change the architecture so that a resource manager or the kernel can allocate devices that preferably exist in the same machines or at least in such a way that the overhead of all the calls would be adequately small.

### **2.2.3 Security**

As described in the overview of the report, Plan9 is a distributed operating system. Therefore the operating system relies on (network) communicating between the (spatially disseminated) computational nodes. The second trade-off mainly discusses the security problems regarding the accessibility of the various servers and their files. This is defined as a trade-off between security versus operability and understandability.

The introduction of the authentication server indeed decreases the operability of the system, because the communication is depending on the authentication. Although it is not clear what happens when the authentication server is absent, one can assume that there will be no communication. Therefore, this method of securing creates a single point of failure in the system. The report does not elaborate on the trade-off between security and understandability.

The report is a bit superficial concerning the authentication. For instance, what will happen to the security of the system when a new computer is added to the network? Or even more important, how do

servers know the true origin of the received messages? It is of course obvious that one should prevent a machine from impersonating the authentication server.

In our opinion, security is actually more than just authenticating users and file permissions. The report however does not elaborate on other security issues or weak spots. For example, it does not discuss or mention the cryptographic protocols, secure sockets layers (SSL) and transport layer security (TLS), which provide communications security. This could (partially) be a solution to the above described questions, but it will also ensure that no one can read data which is intercepted. The use of cryptographic protocols makes the operating system less efficient, due to the resources that are needed to encrypt and decrypt the data and the handshakes between servers and client/terminal and server. The cryptographic keys, which are needed to encrypt and decrypt data, also need to be stored at a central place. It seems logic to assign this task to the authentication agent (factotum), although it is not clear which impact that will have on the security.

## **3 Evaluation**

In this review, we also give a general evaluation of the paper. In the previous sections, the focus was mainly on the trade-offs and architectural changes that are needed for some scenarios (and to which extent the system was able to accommodate such changes). This section concentrates more on the coherence of the paper, the strong and the weak points and the writing style.

### **3.1 Strong points**

The paper is structured very well. The structure starts with an introduction to Plan 9, then a general overview of the system architecture, followed by some quality aspects. Then, the paper focuses on the trade-offs between various quality aspects and finally some conclusions are drawn of the previous chapters. Furthermore, the presentation of the paper is clean and looks professional.

### **3.2 Suggestions for improvements**

The writing style of the paper is not very consistent. Some parts are written using proper English, while other parts could use a significant improvement in spelling and sentence structure.

While the overall structure of the paper is sound and well-thought, the content does not always adhere to the structure. For example, some things are introduced in the conclusions and not earlier (Inferno, which is mentioned very briefly in the introduction and thereafter only in the conclusion). The conclusion should not introduce new topics in the paper, but rather give a short summary of the other sections followed by some concluding thoughts and maybe recommendations for further research. A section that is included in the conclusion is about the future development of Plan 9, but probably this should be moved to a separate section to distinguish it from the real conclusions.

The argumentation behind the trade-offs discussed in the report could use some more elaboration, for example the argumentation why some trade-offs can be seen as a trade-off. The rationale is mentioned very briefly, but the actual trade-off itself is not shown – the authors rather elaborate on the technical aspects of the trade-off that is discussed.

Specifically for the security trade-off: in our opinion, security is taken not broad enough here. The authors discuss the authentication system quite extensively, but security involves more than only authentication: the authors don't show for example how encryption is used in Plan 9 (if any) or how the identity of remote systems is verified.

Finally, we found some parts in the report that contradict each other. For example, the authors mention that Plan 9 is POSIX-compliant using the APE library, but in other parts of the report, the authors conclude that Plan 9 is not POSIX-compliant (or only partially).

## **4 Conclusion**

This report evaluates the review about the architecture and the quality trade-offs of the Plan 9 Operating System. It mainly focuses on the impact that changes in the requirements can have on the architecture of Plan 9, providing at the same time scenarios that support these changes. The easier scenario was to add a simple new hardware type to a Plan 9 installation where we concluded that it does not require big changes from the developers. However the next three scenarios, in which important quality aspects are emphasized like extendibility, performance and security, have big impact in the architecture. The scenarios included a simple change in the existing hardware devices like a mouse with more buttons, a change in the quality requirements so as the system can run applications that are demanding in terms of computing resources, and the third scenario was to introduce cryptographic protocols so as to increase the systems security. In these three scenarios our review focuses on the big range of modifications in the architecture that are needed, for requirements that are taken for-granted nowadays.

Finally, some general remarks are made regarding the content of the reviewed paper. As we can see in the evaluation section, the reviewed document is a well structured profound study on Plan 9, but it needs more consistency and proof reading. Moreover some trade-offs required further explanation, perhaps with use-cases, so as to be more comprehensible to the reader.

## 5 References

Architectural overview of Plan 9 (Stijn van Drongelen, Maria Hernandez, Johannes Leupolz, Alejandro Serrano).

Bohner, S.A. and R.S. Arnold, Eds. (1996). Software Change Impact Analysis. Los Alamitos, California, USA, IEEE Computer Society Press.

Cox, R., Grosse, E., Pike, R., Presotto, D., and Quinlan, S. (2002). Security in Plan 9. Retrieved January 26, 2011 from <http://plan9.bell-labs.com/sys/doc/auth.pdf>

Kilpinen, M.S. (2008). The Emergence of Change at the Systems Engineering and Software Design Interface: An Investigation of Impact Analysis. Retrieved January 25, 2011 from [http://www-edc.eng.cam.ac.uk/research/changemanagement/cm1/softwareystems/kilpinen\\_phd\\_thesis.pdf](http://www-edc.eng.cam.ac.uk/research/changemanagement/cm1/softwareystems/kilpinen_phd_thesis.pdf)

Pike, Presotto, Dorward, Flandrena, Thompson, Trickey and Winterbottom (1995). Plan 9 From Bell Labs. Retrieved January 18, 2011 from <http://plan9.bell-labs.com/sys/doc/9.pdf>