Towards realistic Component Deployment; Understanding the role of implicit assumptions

Gerrit Muller
Philips Research
Prof Holstlaan 4 (WL01)
5656 AA Eindhoven
The Netherlands
gerrit.muller@philips.com

version 0.1

7th October 1999

1 Introduction

The decision to start a component based development strategy is often based on extreme, unrealistic high expectations. Those expectations include:

- No single monolithic release moment, with its related integration and testing time
- Re-use in time and over applications
- Graceful evolution of components
- Easy composition of new functions, services or applications

Hardware components are often used as the analogon of the way software components can be used. This analogon is very good. Unfortunately a very optimistic perception is present about component (re)use, a crititcal attitude is missing during the decision making process.

The root cause of delays and underperformance of component based development is that the majority of managers and designers don't understand the amount of implicit assumptions made during the development of components.

When the expectation level of component based development is reduced to a realistic level, then component based development is a very promising development strategy.

2 Component Based Development

Component based development consolidates know how in a component, which is described by its interface and which is implemented to be able to use it. To enable the composition of an application components adher to a component model, which defines a.o.:

- identification
- instantation
- communication

An important goal persued by component developers is a autonomous lifecycle for each individual component. The decoupling obtianed by teh component model and the strict interface management are the most important tools to enable life cycle decoupling.

The popular view on components is to stress the importance of (stable) interfaces. Classic functional or object oriented programming describes modules by their call or method interface. This interface defines mostly what the module *provides*. On top of this component definitions also describe what the module *requires*, see figure 2. By providing both the *provides* and *requires* interfaces the composition process is enabled, in which multiple components are combined to perform a higher level application.

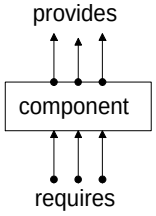

Figure 1: provides and requires component interface model

Composition of easy to use and well performing applications by means of components is much more difficult than most people realize, despite the extensive inteface desription. This is caused by an omission in the *provides* and *requires* interface model:

- the assumptions belonging to the component are missing

Analog to the *provides* and *requires* interface two types of assumptions are important:

- How will the component be used?
- Which constraints or limitations are present in the components which are used by this component?

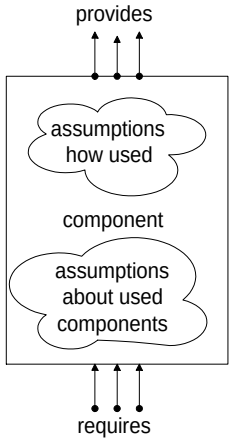

Figure 2: Component assumptions

3 Example medical viewing components

Figure 3 shows an example of composing viewers for different applications by means of components. Four applications are shown:

- Ultra sound
- Cardiac
- MRI volume
- simple X-ray

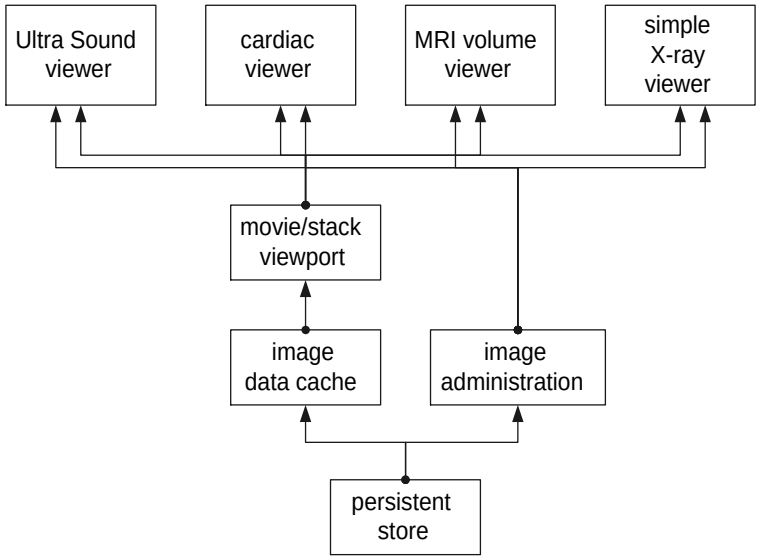

Figure 3: Example of components used in medical viewing applications

All these applications need a component capable of showing a stack or a sequence of images, with manipulation capabilities for the user. In this example this component is named "stack viewer". The patient and image data is managed by a component called "image administration", which provide query, search, selection, update etcetera functionality. Two more "infrastructural" components are shown, the image data cache, providing fast access to large image data sets and the persistent store, providing reliable and robust storage persistently.

In real life many more components will be present, such as communication with the outside world, storage on other media, printing on film. For sake of simplicity these components are left out of the example.

Table 1 gives an overview of the typical characteristics of each application. All these applications have in common that the radiologist does not want to spent time on the viewing. In the majority of cases the diagnostic viewing is less than 1 minute.

Table 1: Typical characteristics of every application

application	image resolution	# bits	images per series	dimensions
Ultra Sound	256^2	8	1000	t
Cardiac	1024^2	8..10	200	t
MRI Volume	$256^2..1024^2$	8..12	20..500	z, t, other
simple X-ray	$1k^2..4k^2$	8..12	1..10	t

Every application makes assumptions about the components being used, for instance about:

- the viewing performance of the movie/stack viewport
- the access (search or sort) performance of the image administration
- the update performance of movie/stack viewport and the image administration

The movie/stack viewer component and the image administration component will be designed with one or more certain applications in mind. These components make assumptions about their deployment, for instance about:

- the granularity of search, sort and selection access to the image administration
- the functionality and performance of navigation through a series of images
- the required quality and perception of displayed images

The values given in table 1 show the variation of a number of key parameters for these different applications. Most of these parameters have a significant impact on the specification assumptions

4 Design of a new component

4.1 Assumptions how the component will be deployed

A new component will be made with a given purpose in mind. In the ideal world the application domain is explored before a new component is developed. The result of such an exploration is a scope definition for the component development.

In the next stage it is recommended to select a limited scope to develop a first version of a component. The purpose of this first developement is to get feedback from actual deployment. A limited scope enables a faster implementation resulting in earlier feedback.

In the medical example the exploration could include the simple X-ray, the cardiac and the MRI volume viewers, while in the next stage the simple X-ray viewer is used as carrier for the first developement.

In this approach it is clear that the characteristics of the later deployment are used in the design of a new component. It will be good in the applications for which it is designed, while it ought to be robust for deviating applications.

4.2 Assumptions about other components to be used

During the feasibility stage of the development of a new component the critical characteristics will be measured of the compenents which will be used. These characteristics will be used in the design of the new component[1]. For instance in the applications in the medical example if single image updates consume too much time, than the updates will be combined or performed in the background.

The know how of the components being used is in this way "built-in" in the application components. Some people will argue this to be bad practice. However the "real world" is full of constraints and limitations. Most users will prefer a well performing product above a product which is frustratingly slow, but which is internally pure of design[2].

Some of these trade-offs will be documented in the design and thereby make the assumptions explicit. However many trade-offs are made unconsciuosly (good designers make them intuitively) and stay implicit.

[1] Less experienced developers have the bad habit of omitting this fact finding stage, often resulting in a dramatic bad performance of the final product.

[2] This trade-off is clearly a function of time. As soon as the performance of the underlying system is sufficient to support the pure design, the purity will pay off for certain, by more flexibility, a greater operating range etcetera.

5 Conclusions

History
Version 0, august 12th 1999
Under construction, no change control yet