The Virtual Lab Abstract Machine

Adam Belloum David Groep

document version 18th April 2001

$Id: amrequire.tex,v 1.2 2000/12/04 15:38:08 adam Exp $

These are simple thoughts, some of what is written, or suggested is obvious and may be well known within the computer science community, the aim is not to bring something new, it is more towards pointing out important points for the design, identifying challenges, and foreseeing problems.

Part I
An Approach for the Design

It is recommended in a number of studies [5, 3, 2] that software architecture should be viewed and described form different perspectives and should identify its components, their static inter-relationship, their dynamic interactions, properties and characteristics and constraints on these item (a consensus reached in the Process-sensitive SEE Architecture workshop held in 1992). It is thus very important for us to define the elements of the architecture we are aiming for.

- Because we are following a case-study driven approach, the components of our design strongly depend on these case-studies. The rest of the elements of the design are related to these components. In the context of the VLAM-G, these components are called *Generic Functions*.
- The *Generic Functions* are still not well defined, as first guess we can think of some standard data filters, some visualisation tools, simulation models etc. It might be too early to define to complete set of this generic functions, but still it might be a good time to define constraints for them. Defining constraints will simplify what is known in software design as 'Software composing' [1] and avoid the 'Architectural Mismatch' [4].

1 An approach for the design of the Abstract Machine

Designing a software architecture involves taking care of many different aspects, including a structural view, a behavioral view and a resource usage view [5, 3]. It has been agreed on in early stages of the Abstract Machine design not to start from scratch, but to try to make use of existing systems, if they comply to the requirements defined for the VLAM-G. It has also been agreed on to follow a case-study driven approach to define some components of the design. With regard to the traditional phases recommended in software design, our approach is different in certain points:

1. Make use of existing systems instead of developing: this implies we should study first existing systems and compare them, based on what is needed in the VLAM-G.

   It is almost impossible to find an existing system that satisfies our needs exactly. We think the target should be to find the one that satisfies most of VLAM-G needs. In taking the decision, of course not all the requirements have the same "importance" in the Design of the Abstract Machine. Important questions has to be asked to guide the selection of the existing systems:

   - What should be looked for in existing system, and what should be developed in the VLAM-G?
   - What are the criteria to sort the list of requirements list?

2. Allow the users to define and 'implement' the components of the Abstract Machine.

   The case-driven approach we are following has the advantage to end up with a very wide variety of Generic Functions. It might be, however, that these functions are not generic enough to allow easy reuse, since they were designed by people primarily interested in their own case-study. The role of the team working on the design of the Abstract Machine is to make sure that components will be designed in a way that eases the final target: increase of reuse. Among other points that have to be watched during the design of the generic functions are:

   - providing more than one interface
   - providing standard mapping functions for certain class of data
   - using standard types *????*
   - Testability
   - Maintenance
   - performance
   - reliability
   - portability

1.1 What to do Next in Abstract Machine Design

T1 **Define de VLAM-G requirements** a still incomplete list of requirement is attached as part of this document for discussion. Having such a list of requirements is important for many ongoing tasks:

- Selection of existing systems to be considered for the study.
- Comparison of the selected existing systems.
- Can be used as guideline for the design and the implementation of the generic functions.

T2 **Define the constraints for the generic functions**

T3 **Define a basic set of generic functions**

T4 **Comparing the different systems that have been proposed so far**

- re-implementing on top of Globus
- Manifold

- HLA *???*
- Splice

Some of tasks listed above have to be done in collaboration with the persons in charge of the case-studies (these are T1, T2, and T3). However, task T4 interests only the team in charge of the Abstract Machine design and implementation.

Part II
Abstract Machine Requirements

The functionality of the Virtual Lab can be divided in two main parts: *(i)* the fundamental operations or functionalities and *(ii)* a VLAM-G proper that is concerned with linking the functionalities provided into 'experiments'. This draft document is a first step towards defining the requirements for this VLAM-G proper.

In Fig. 1 an attempt of a very schematic overview of the (entire) Virtual Lab is given. The fundamental operations provide atomic functionality and can be described as 'modules' to be re-used in various experiments. In this way, the interdependency of the modules is minimized and most, if not all, of them can be developed separately and in parallel.

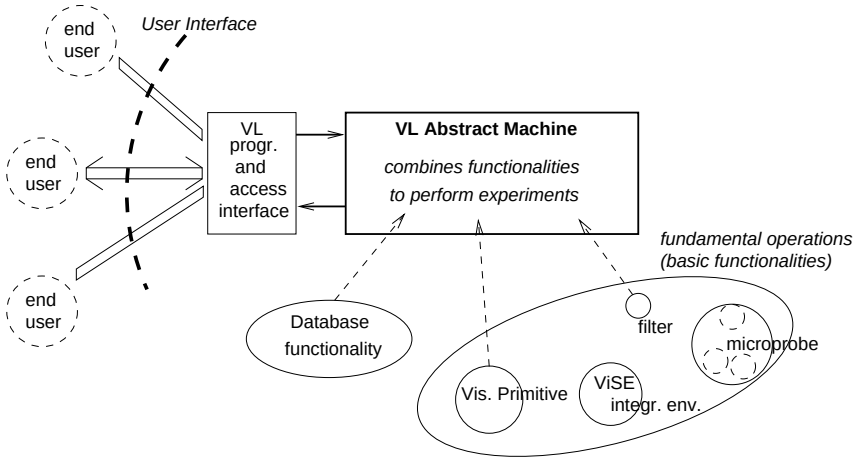

Figure 1: *Schematic overview of a potential Virtual Lab, indicating an abstract machine and a VLAM-G programming and access interface as part of the VLAM-G proper. The basic operations and the database (a potentially special kind of functionality) are provided as atomic entities to the abstract machine.*

2 Architecture

Four classes of requirements can be defined:

- Functional
- Performance
- Fault tolerance
- Security

2.1 Functional requirements

Fu.1 *the same modules can be connected in various patterns*
This allows modules to be re-used in several experiments without the need for changes in said module.

Fu.2 *modules can be added to a running experiment*
In this way new end-users can 'connect' to a running experiment and add modules catering their specific (viewing) needs. Also end-users might be (partially) modelled as new participating modules.

Fu.3 *modules may be composed of smaller modules in a 'recursive' way*
Higher-level modules may be best represented as a conglomerate of smaller modules (*e.g.*, micro probe → beam-steering + acquisition + microscopy + target movement). During the life-time of the VLAM-G, implementation of a function may change, from an atomic entity into such a conglomerate (to re-use functionality) or back (to improve performance). It is nice to have a scheme where such a change does not modify the experiment as a whole.

2.2 Performance requirements

P.1 *Resource management*
The availability of machines and network bandwidth may vary overtime. The abstract machine is built on such an ever-changing environment. Besides, load balancing of computing nodes and the minimizing of network load by optimizing the topology of an experiment can be used to optimize performance.

P.2 *provide high-throughput connections between modules*
Multiple data flows in a running experiment with sustained data rates between 1MB/s (micro probe) and 30 MB/s (medical imaging) are to be expected. Moreover having more than one destination for such a data stream will be common.

2.3 Fault tolerance requirements

Ft.1 *coping with spontaneously disappearing resources*
At least at the global level, network disruption and module-local failures should not halt the VLAM-G or influence other experiments. It might be worthwhile to extended this to single experiments as well.

Ft.2 *restrict connection of modules to matching or equivalent data types*
This will prevent run-time failure of modules due to bad input types and might eliminate the need for continuous type-checking within each module. Also automatic selection of 'conversion agents' or module chains may be incorporated to prevent type mismatches.

2.4 Security and accounting

S.1 *secure communication between VLAM-G experiment parts and among end-users*
Make spoofing and eavesdropping impossible by using encryption techniques, *e.g.*, Secure Sockets (SSL) based on public-key encryption.

S.2 *user authorisation should be secure*
Identity verification depends on method of access, *e.g.*, for web-based access secure HTTPS access (X.509 certificates).

S.3 *access restrictions*
Resources may be off-limits to certain end-users. Access control lists (per user of per group of users) might be used to determine access to resources on global level and inside experiments.

S.4 *auditing and accounting*
Commercial operation will need accounting of module/functionality use. Based on module type/duration of usage or on amount of data transferred. Ways of collecting these data and storing them in a database should be devised.

3 Data domains

Data exchange among the various modules and experiments in the VLAM-G requires a detailed description of the data types produced and consumed by the various modules.

D.1 *data 'domains' have a precise definition*
Since only identical or equivalent data types can be transported between modules, authors of the module specifications should clearly state the required input and output data formats. These data types may be organised in a hierarchical manner, such that general-purpose modules may replace more specialised modules.
Connections between modules should be restricted to compatible or equivalent types only.

D.2 *Conversion modules between data-domains will be needed*
Application domain standards will need to be catered for (by the application domain developers!). The proper 'chain' of conversion modules may be auto-selected for (or auto-proposed to) the end-user by agents.

D.3 *Use as few distinct data types as reasonable achievable*

D.4 *Should a 'database module' be capable of storing all data sorts transported within the VLAM-G?*

4 User Interface

For now some random thoughts on the user interface requirements:

UI.1 *Designing and experiment modification by means of drag&drop of modules and connections*
Provides an understandable interface for power-users and end-users alike. Visual representation of the running experiment allows easy on-the-fly changes in experiment and data-flow layout. Outgoing connections are multiplexed as necessary.

UI.2 *should more than one user be able to modify the experimental setup?*
The 'initiator' of the experiment might leave the experiment and delegate control to one or more other users. Experiments are not 'owned' by a specific end-user.

UI.3 *'specialized' views of the experiment per end-user*
End-users have a restricted view on the experiment and see only the 'global' modules and their own 'additions'. In particular, so that they do not get confused with the modules added by other end-users and also are not able to interfere with these modules.

UI.4 *can 'experiment view' and per-module configurable settings (config) be integrated*
Configuration settings are generally localized per-module. Selecting a module for config may provide the user with a control window or, if the module itself is composed of several modules, a new 'experiment view'.

UI.5 *End-user interface may support various connection types*
Besides web-based access, more restricted interfaces (text-only, maybe voice/touch tone access) should be possible (may later be extended to WAP/SMS/etc. interfaces). Out-bound-only needs for voice/SMS/e-mail messages for notification and special events.

UI.6 *seamless integration of conferencing?*
Existing tools like SunForum or NetMeeting are already available and may potentially be integrated. Whether integration can be at the user-interface level only remains to be seen.

5 Mid-term Prototype

References

[1] A. Abd-Allah and B. Boehm. Reasoning about the composition of heterogeneous architecture. Technical Report UCS-CSE-95-503, University of Southern California, 1995.

[2] A. Abd-Allah and B. Boehm. Models for computing heterogeneous software architecture. Technical Report UCS-CSE-96-505, University of Southern California, 1996.

[3] C. Gacek, A. Abd-Allah, B. Clerk, and B. Boehm. On the definition of the software system architecture. Technical Report UCS-CSE-95-500, University of Southern California, 1995.

[4] D. Garlan, R. Allen, and J. Ockerbloom. Architectural mismatch: Why reuse is so hard. *IEEE software*, Vol. 12(No. 6), November 1995.

[5] J. R. Putman. Architecture views in creating and using software systems architecture for large scale systems. Technical report, MITRE Corporation Bedford, 1995.