Enhancements to the XAL Online Model for J-PARC **Topics in Software Engineering** and RMS Envelope Simulation

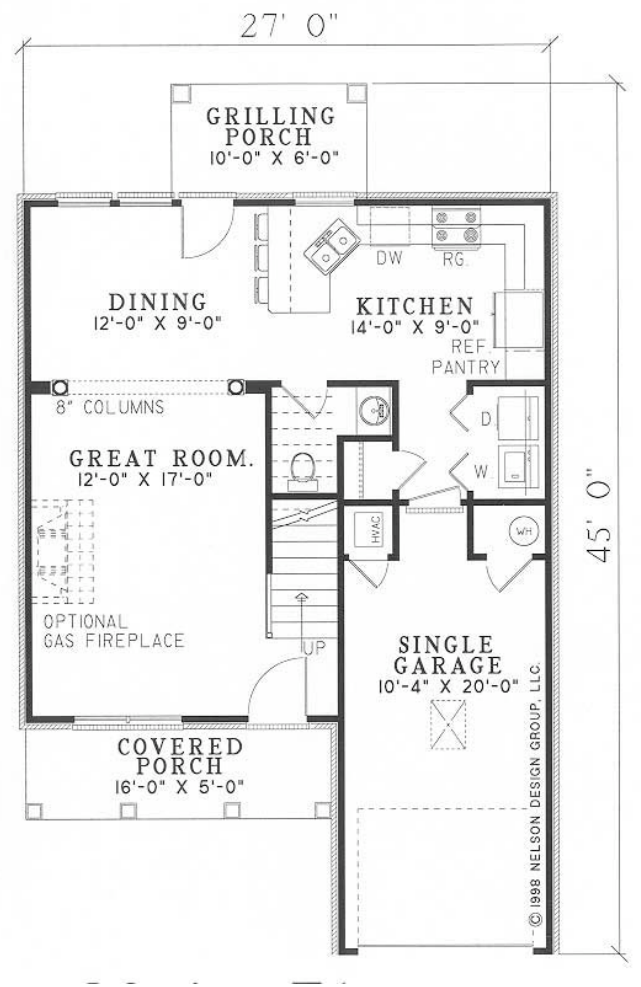
Christopher K. Allen
Los Alamos National Laboratory
Los Alamos, New Mexico

Abstract

The XAL accelerator application framework was originally designed with specific architectural goals which are important to recognize in order that future upgrades are consistent with these goals. Consequently software engineering is a very important aspect of XAL development. **I will discuss various topics of software engineering in general, and with specific regard to the XAL framework.** Also, I will briefly outline the specific enhancements to the XAL online model made during my sabbatical time at J-PARC. Some of this work included adding additional simulation capabilities to the online model, correcting existing ones, and verification. However, much effort was also devoted toward refactoring existing code into a more robust and upgradeable software system consistent with the XAL architecture.

Outline

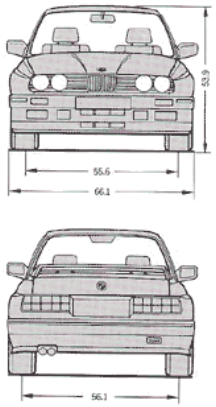
1. Motivation
2. Managing Software Development
3. Engineering “Rules of Thumb”
4. Accelerators and Software Engineering
5. Summary

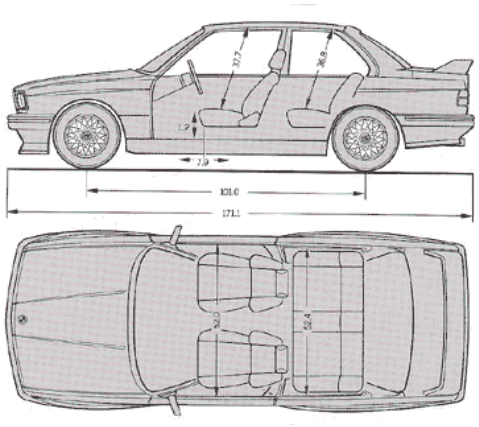


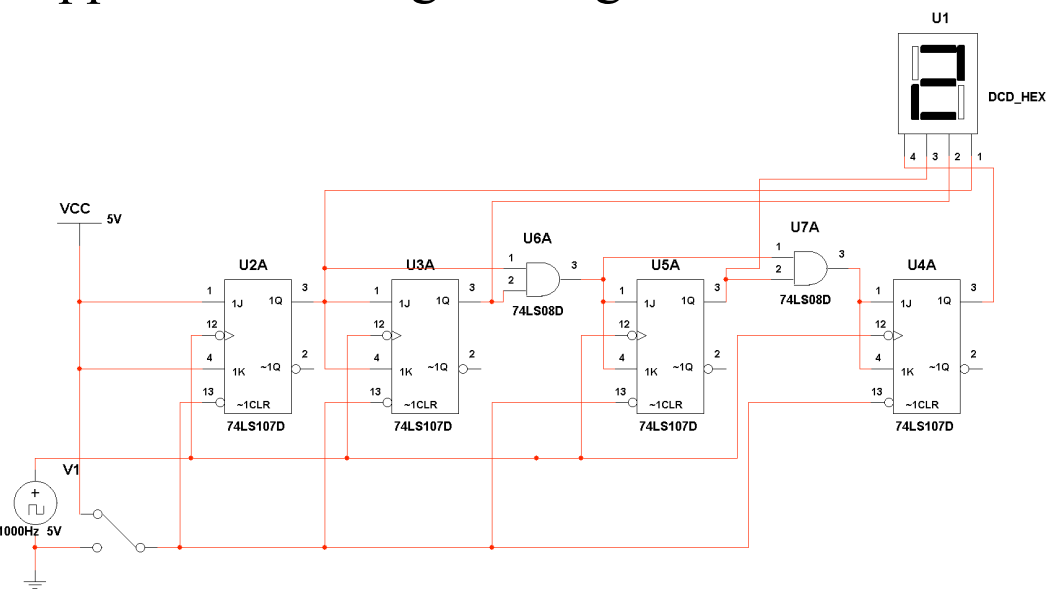
Software Engineering
Definition

According to the IEEE

> "The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software: that is, the application of engineering to software."







Software Engineering
1. Motivation

- Complex systems are best implemented with engineering
  - Would you build a house without blueprints?
  - Would you build **your** house without blueprints?

- Since software is intangible (practically invisible) it is more susceptible to neglect in this area
  - This is a dangerous trap
  - Many shortcomings are easily obfuscated
  - These shortcomings almost always become crises later on

Software Engineering

1. Motivation

Software Engineering
1. Motivation

The moral of the story is that your going to put your time in

1. Do you want to spend time at the beginning of the project designing, documenting, and quantifying?
   (not so sexy)

2. Or do you want to spend your time at the end putting out fires?
   (even less sexy)

Software Engineering
2. Managing Software Development

Software engineering can make management nervous

This is understandable, it is relatively new

- Few metrics to gauge progress
- Design phase
  - No code is being written
  - Typically very long
- Innovation is hard to manage
  - Many unknowns
- “I don’t care what it looks like on the inside”
  - As long as it works “who cares”

Software Engineering
2. Managing Software Development

Let us address each of these issues

- Progress Metrics
- Design phase
- Innovation
- “Who cares”

=>Managing Risk

2. Managing Software Development

i. Metrics

Old software progress metrics are almost meaningless

Imagine building a house without a design

- "I've used 1,000 board-ft of lumber – I'm half finished!

Imagine building a large software system without a design

- "I've written 100,000 lines of code – I'm half finished!"

- Ironically, old progress metric was *lines of code* written.

- A more meaningful progress metric is *tasks completed*
  - Must identify tasks (engineering)

2. Managing Software Development
ii. Design

Eight Phases of Development (WWISA)

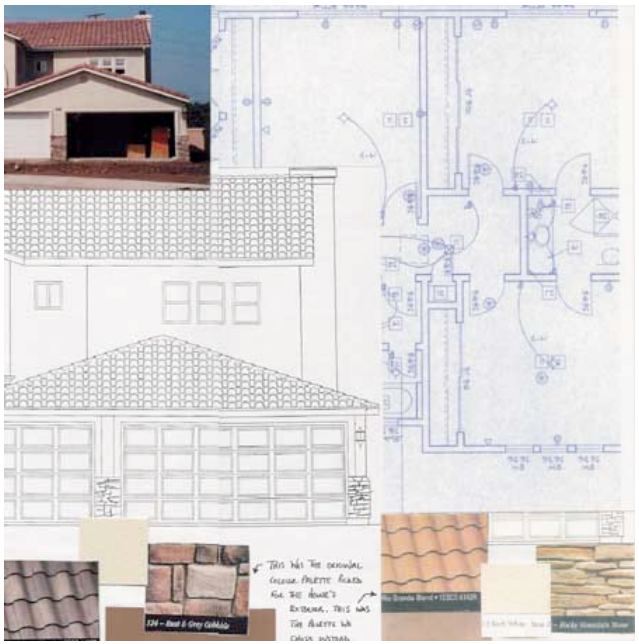
1. Pre-design – scope, requirements, expectation
2. Domain analysis – document system behavior
3. Schematic design – architectural level design
4. Design development – detailed design
5. Project documents – construction process detail
6. Staffing or contracting – personnel, costing, etc.
7. Construction – software implementation
8. Post-construction – deployment, maintenance, etc.

2. Managing Software Development
ii. Design

Note that most effort is devoted toward design, documentation, costing, etc.

- Only about 15-30% of the effort is implementation.

- However, it is **easy and fast** to implement large blocks of code if blueprints already exist.
  - (The thinking part is hard)



2. Managing Software Development
ii. Design

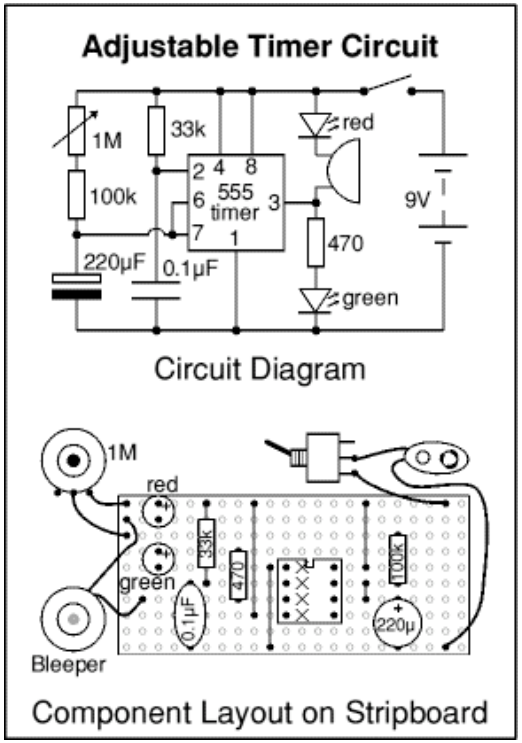
My contention that design has always occupied most of the effort

- Previously the design and implementation phase were not separate
- Coding and designing at the same time
  - Leads to a “wandering” style of development
  - No meaningful metrics for progress, operation, etc.
  - Does not support team development

2. Managing Software Development
iii. Innovation

As developers, our jobs are complicated by the ethereal nature of software

- Counter-example: mechanical engineers can **see**
  - A bad design
  - A flawed prototype
  - An implementation "bug"
- Electrical engineers were faced with a similar problem
  - Developed circuit diagrams
  - Circuit simulators
- Software engineers now have similar tools



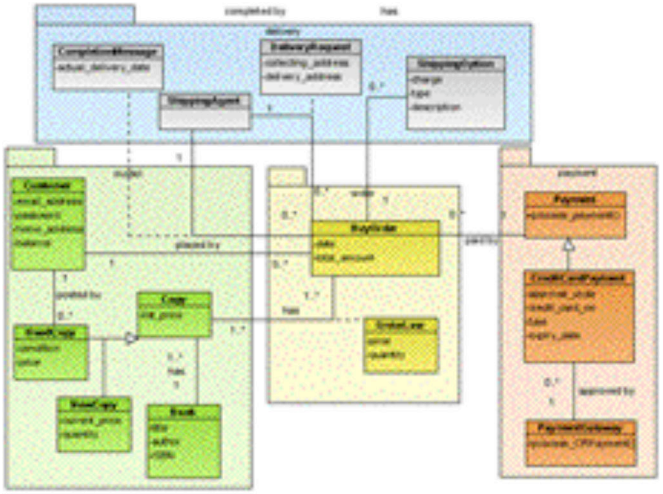
2. Managing Software Development
iii. Innovation

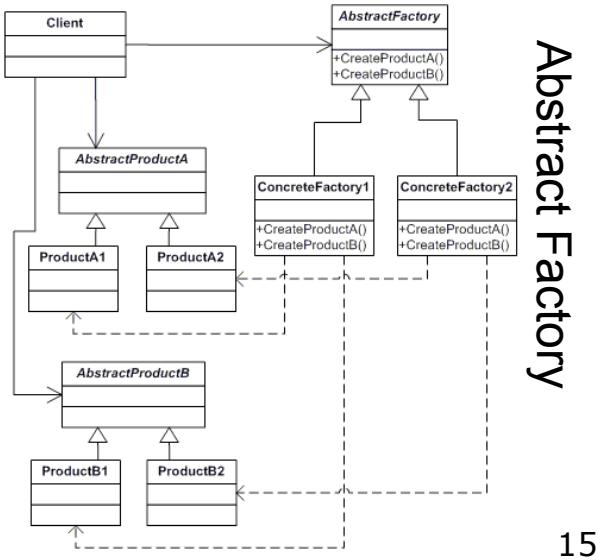
Software Engineering Tools

- UML is a formalized language for blueprinting software.
  - Analogous to circuit diagrams or mechanical drawings
  - There are many commercial UML tools
    - Rational Rose

- Design Patterns
  - Common architectural solutions to common engineering tasks
  - Analogous to amplifiers, DSPs, A/D converters, etc.

Such tools help us develop an *a priori* appreciation for the difficulty of a software task





Abstract Factory

2. Managing Software Development

IV. “Who cares… as long as it works”

Would you buy the car without having a look under the hood?

It works fine now, but wait until you drive it off the lot…

Proverbial “barbeque under the hood”

2. Managing Software Development

IV. “Who cares what’s under the hood”

Brittle Code

- It works **now** and it is quick to write
- But it breaks often, and a significant part of development is spent fixing it.
  - It is hard to understand
  - It is hard to maintain
  - It is hard to upgrade
- It typically requires much more time and effort in the **long term**

“I hate changing to Daylight Saving Time.”

Software Engineering
3. Rules of Thumb

- What is good software engineering?

Good software engineering and pornography
"I can't define it but I know it when I see it."

Software Engineering
3. Rules of Thumb

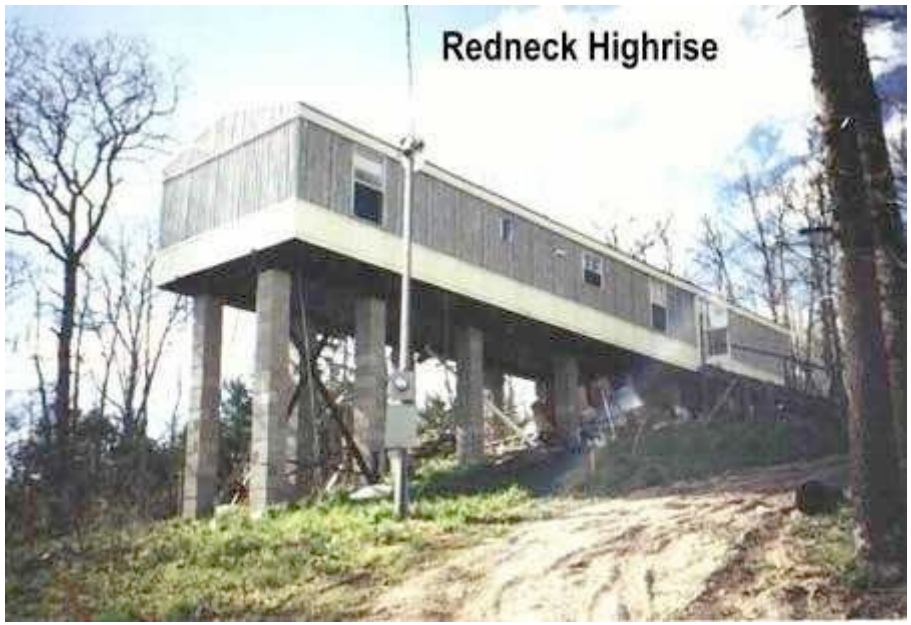
Design your software as robust as possible

- Your entire software life-cycle is affected by what you do at the design stage
- You cannot anticipate everything, but neglect here will certainly cost you throughout the software lifetime

Software Engineering
3. Rules of Thumb

Is your software upgradeable?

- There will always be upgrades!
- Your foundation must be solid and able to accommodate the future



Software Engineering
3. Rules of Thumb

Recognize the "Death March"

There are ways to survive a death march

- see E. Yourdon "Death March"
- Planning is essential here
- Natural tendency is to jump immediately into implementation
  - Worst strategy

"I want you to draw up plans
for a city that can be built in a day."

Software Engineering
3. Rules of Thumb

Spend the time to write clear, concise, documented code

- If it is only you on a project, and will only be you, forever, then do whatever you want, because the rest of us won't look at it.

- However, if the project involves other people, requires production quality, and expects future modifications and growth, by all means please heed the principles of software engineering.

Software Engineering
3. Rules of Thumb

- For example, do you want to deal with this?

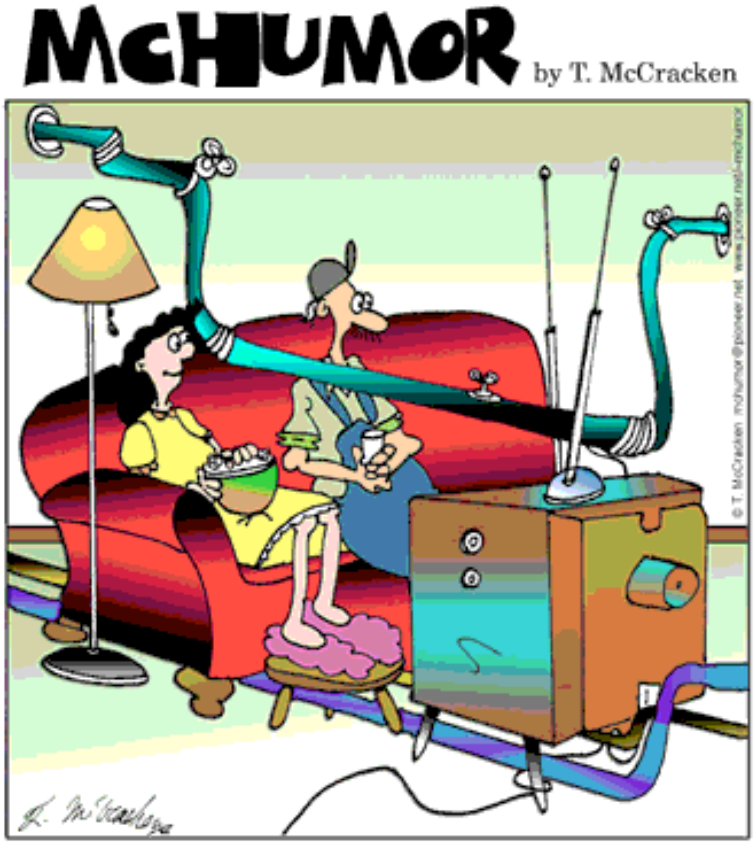
A 2004 Winner of the International Obfuscated C Code Contest (IOCCC)
(Polynomial Graphing Program)

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define _                       ;double
#define void                            x,x
#define case(break,default)     break[O]:default[O]:
#define switch(bool)            ;for(;x<bool;
#define do(if,else)             inIine(else)>int##if?
#define true                            (--void++)
#define false                           (++void--)

char*O=" <60>!?\\\n"_ doubIe[010]_ int0,int1 _ Iong=0 _ inIine(int eIse){int
O1O=!O _ I=!O;for(;O1O<010;++O1O)I+=(O1O[doubIe]*pow(eIse,O1O));return I;}int
main(int booI,char*eIse[]){int I=1,x=-*O;if(eIse){for(;I<010+1;I++)I[doubIe-1]
=booI>I?atof(I[eIse]):!O switch(*O)x++)abs(inIine(x))>Iong&&(Iong=abs(inIine(x
)));int1=Iong;main(-*O>>1,0);}else{if(booI<*O>>1){int0=int1;int1=int0-2*Iong/0
[O]switch(5[O]))putchar(x-*O?(int0>=inIine(x)&&do(1,x)do(0,true)do(0,false)
case(2,1)do(1,true)do(0,false)6[O]case(-3,6)do(0,false)6[O]-3[O]:do(1,false)
case(5,4)x?booI?0:6[O]:7[O])+*O:8[O]),x++;main(++booI,0);}}}
```

Software Engineering
3. Rules of Thumb

Do not put code somewhere just because it is convenient

- This is faster in the beginning but costs a lot of time later
- E.G., Don't run the sewer line through the heating duct just because the hole is already there.



Why it's bad when home owners change their minds about the bathroom's location late in a building project.

Software Engineering
3. Rules of Thumb

Avoid "Over Engineering"

- Scale your engineering efforts to the size of the project
  - A dog house
  - The White House
- "Paralysis by Analysis"
  - When a clear path is not apparent, you may need instinct

"Sounds better than live?"

Software Engineering
3. Rules of Thumb

Knowing when to quit

- Sometimes legacy code can no longer be salvaged
- Move away from functional programming
  - Brittle code
- Stop using FORTRAN
  - Does not support engineering

Need for Software Engineering

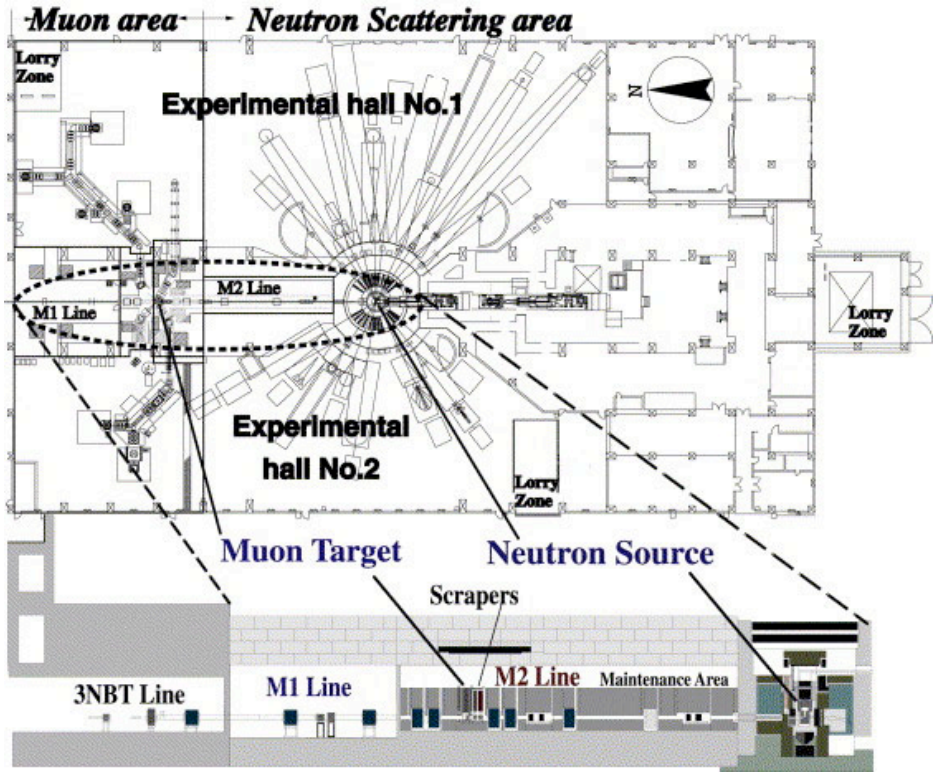
4. Software Engineering and Accelerators

Some of the worst software-bugs which ever occurred:

- 1985: 6 People died due to radiation overdose in the Therac-25 X-ray therapy apparatus caused by a software bug.
- 1996: An Ariane 5 spacecraft exploded 36 seconds after take-off. The problem was software originally used in its predecessor, Ariane 4. A directional correction too large for the new spacecraft was applied, exceeding aerodynamic tolerances.
- 1999: The Mars Climate Orbiter incinerated in the Martian atmosphere because data that was in expressed in English units was entered into software designed for metric units.
- The original software in the F-16 fighter jet would have turned the plane upside-down when it crossed the equator. Fortunately, this problem was detected early via simulation.

4. Software Engineering and Accelerators

- Accelerators are (arguably?) complex systems
  - More complicated than cars…
  - More complicated than houses…
- Then why would not an accelerator control and/or simulation code not also be complex?
  - It is unreasonable to think that the construction of such systems would not require a significant engineering effort



Software Engineering and Accelerators
There is a Need

The newest machines are the most sophisticated ever built

- Highest power
- Tightest tolerances
- Largest scales
- Multiple applications

- We need sophisticated software that can analyze and control these machines
  - Software simulations
    - Not just a bunch of calculations
    - Put together the model in a logical manner
  - Feedback control
  - **Modern control**

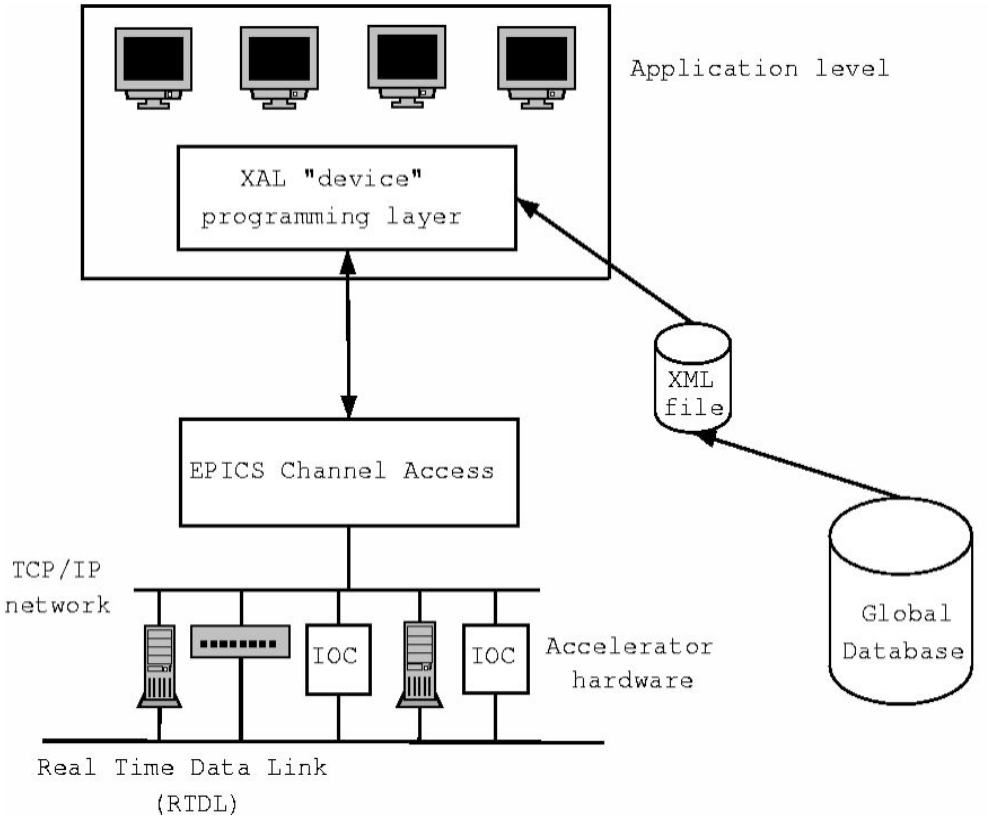
- Bottom Line – Need to implement control and simulation software that is
    - Reliability!
    - Maintainability!
    - Upgradeability!

Example: Accelerators and Software Engineering

XAL

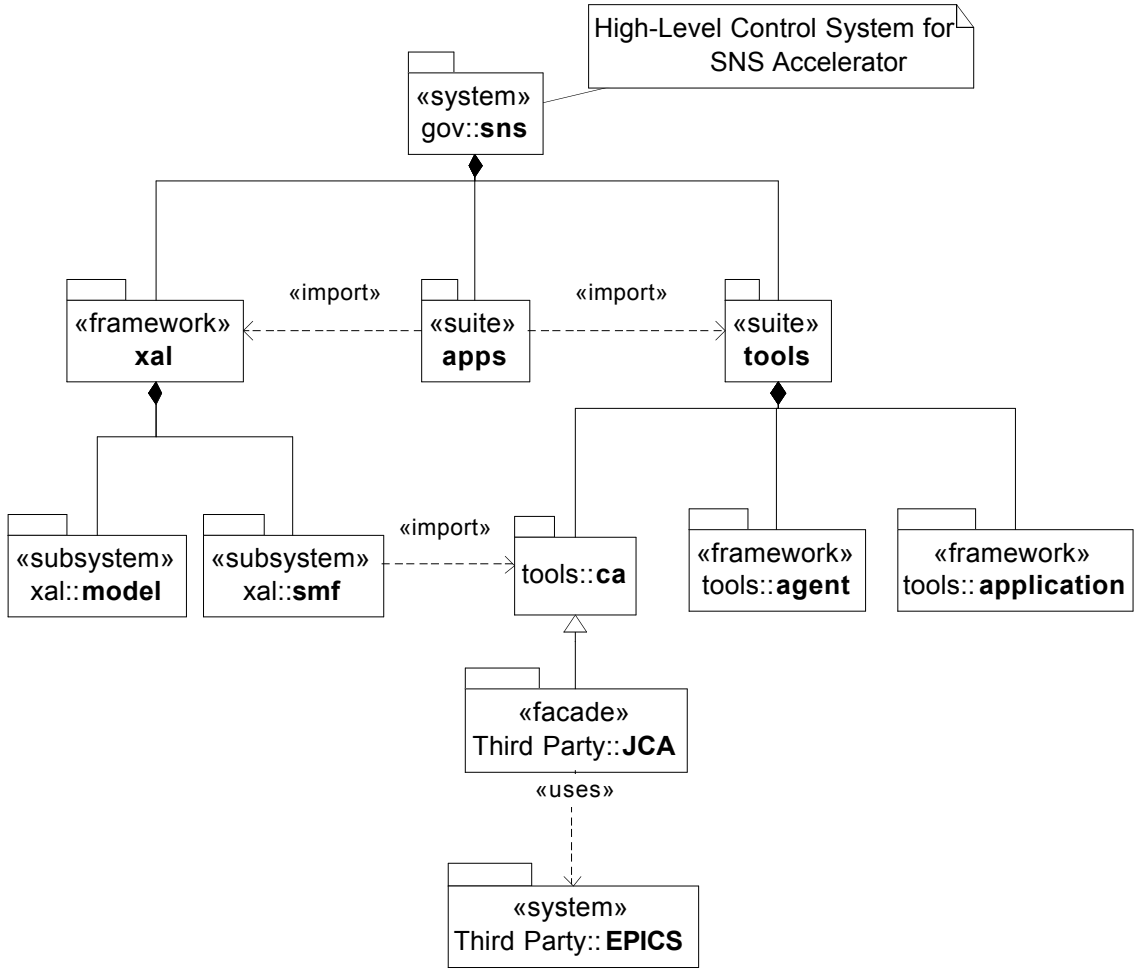
A Framework for Portable High-Level Control of Charged Particle Accelerators

Conceptual (Pre-Design)

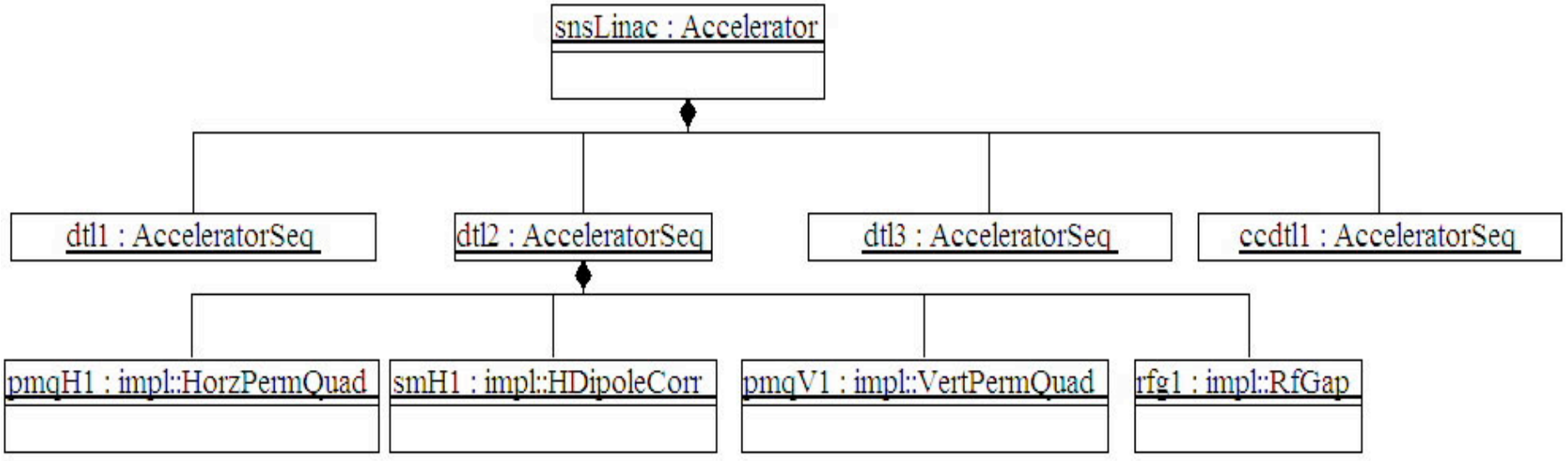


XAL Architecture

Subsystem Diagrams

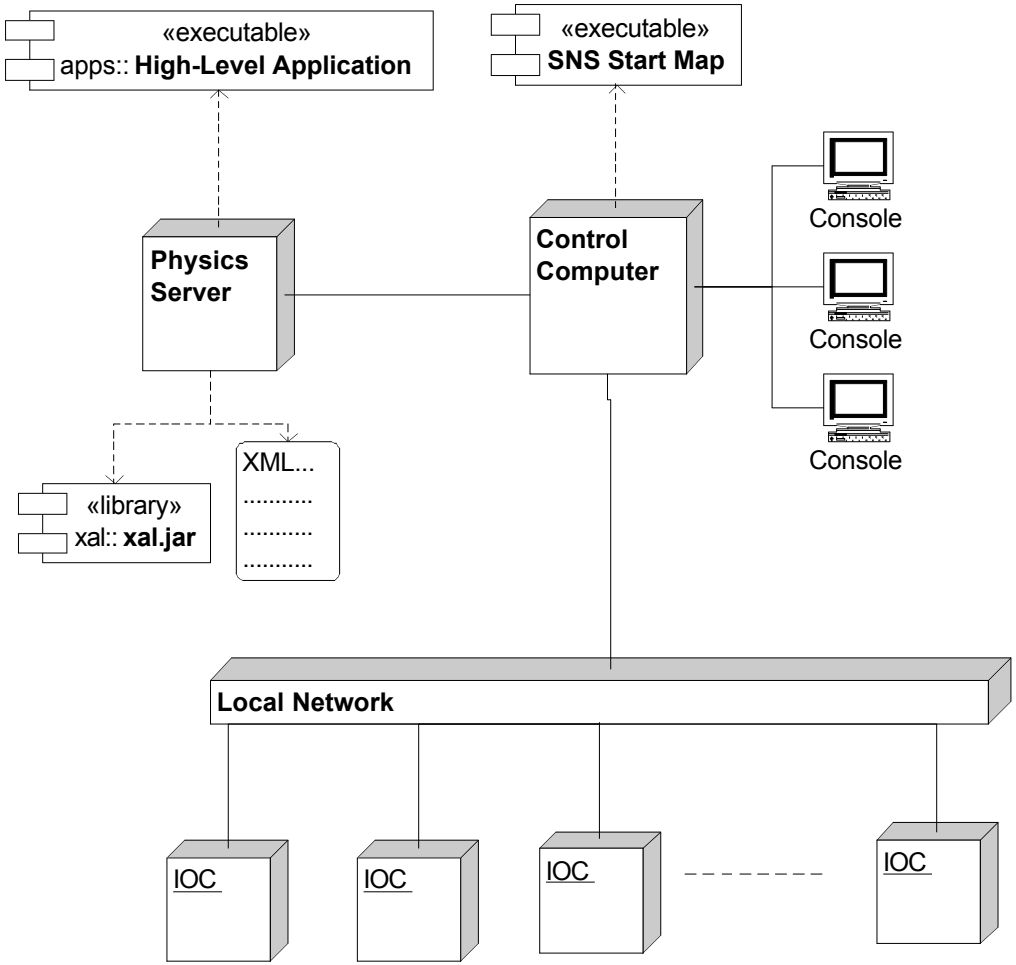


XAL Architecture
Class Diagrams

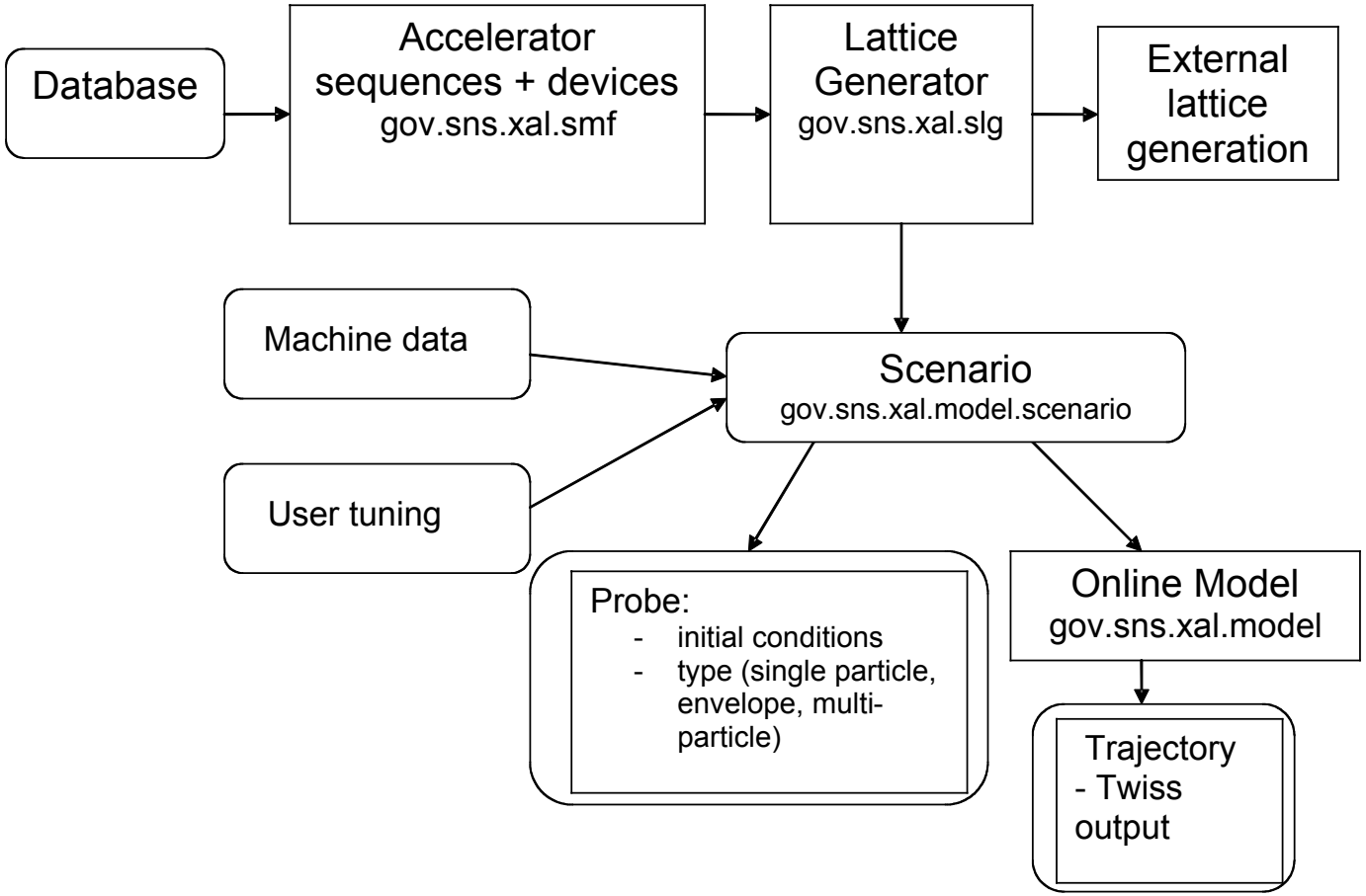


XAL Architecture

Deployment Diagram



XAL Architecture
Communication Diagrams



5. Summary

Accelerators are very complex systems. The operation and simulation of these complicated and expensive machines rely upon software, complex software. It is only natural to allocate a significant part of the software development effort to guarantee that this software is reliable, maintainable, and easily upgraded. In other words, in the long term, to provide the most efficient, cost effective, and reliable operating environment for our accelerator complex, we should ensure that this software is engineered to the best available standards.

どうもありがとうございます

皆様

この二年間皆さんと一緒に仕事が出来てうれしかったです。日本では貴重な体験をたくさんさせていただきました。このようなすばらしい機会を与えて下さり本当にありがとうございました。

XAL Architecture
Points to Remember

- XAL Framework
  - Be careful here!
    - Do not make modifications just because it makes your life more convenient
    - Consequences affect **everyone**
- Applications
  - Architecture and engineering are less important
    - However, if you build a general, well-designed application, then everyone can use it

XAL Architecture
Points to Remember

- The lattice file represents the hardware
  - It is **not** a data file
    - Design values only
    - Not for machine parameters setting
- Online Model
  - Three independent components
  - Element/Algorithm/Probe Architecture
    - Only the Algorithm object knows about Elements and Probes

Software Engineering
Rules of Thumb

- Document your code (Javadoc!) (not sexy)
  - It is certainly faster for you **now** to skip documentation
  - But much slower for your **coworkers** later
  - Much slower for **you** later when you forget what you were doing
  - It is also dangerous to skip documentation
    - I usually read the instructions before firing up the bulldozer

Software Engineering
Rules of Thumb

- New software technology
  - Don't use Version 1 products
    - "Microsoft: Where quality is number 1.1"
  - There is always a lot of new software
    - Try to maintain awareness of new technologies and what they do so that you may learn them and use them when needed
  - Before writing new software – Check the Internet
    - For Java this is especially true – and easy

Software Engineering
3. Rules of Thumb

- Software conventions are typically good – use them!
  - Saves much time for the person behind you
  - Why is there always a light switch, inside the door, opposite side, at hand level?

Software Engineering Rules of Thumb

- For proof of principle, quick prototyping, etc., Matlab, Mathematica, Scripting, etc., are all fine.
- However, when it comes time to build a solid software system, a high-level language is more appropriate
  - Supports necessary engineering and architectural constructs
    - Classes, interfaces, components, etc.
  - Supports maintenance and testing
  - Clarity and documentation

- Management styles
  - Maintenance
    - Anticipate problems and prepare for them
  - Crisis intervention
    - Respond to crises as the occur and fix them
- Good software engineering favors the former
  - Design!
    - Reliability
    - Maintainability
    - Upgradeability

Need for Software Engineering

4. Software Engineering and Accelerators

Some situations are obvious

Many applications require robust, large, and/or complex software systems and, consequently, must be engineered

- Medical applications
  - AED (automated external defibrillator)
  - X-ray equipment
- Aerospace
  - Navigation
  - Automation
- Accelerator control?