

ECOOP Workshop on Programming Languages and Operating Systems (ECOOP-PLOS 2004)

Results of Discussion Groups

Workshop Organizers:

Olaf Spinczyk, University of Erlangen-Nuremberg, Germany

Andreas Gal, University of California, Irvine, USA

Michael Schoettner, University of Ulm, Germany

1. Aspect-Oriented Programming and Operating Systems

Q1) What is the real use of AOP in an OS?

- Same goals as in applications.
- Modularization.
- Crosscutting concerns: policies, synchronization, protection (single vs. multi-user).
- OS are usually more configurable than apps,
- Usually more crosscutting concerns in OS than in apps.

Q2) What problem does it address that can not be solved with a more traditional programming language technique?

- Crosscutting concerns.

Q3) Why not have the possibility to simply change the definition of functions (e.g., copy the definition and insert the advice), and possibly invoke the existing version?

- Great idea.
- We just want a tool that does it!

Q4) Is profiling the killer application for dynamic AOP. Are there others?

- No. Yes.
- Everybody agrees that dynamically loadable modules are a good thing.
- What is their killer application?

Q5) How feasible is dynamic weaving in static languages (performance impact)?

- Doing something dynamically is always expensive.
- The idea is to do it only if you really need it.
- "Do as much statically as possible".

Q6) Why not recompile applications for profiling (especially for embedded systems)?

- See above.
- Server-side weaving was suggested for this.

Q7) What other approaches are possible (for example instrumentation by the VM)?

- It's the idea that counts.
- You mean, what other implementations are possible?
-

Q8) How difficult is it to understand and teach AOP?

- Not at all.
- It's easier than OO.

Q9) Why not weaving into the application code instead of the OS code (for example when monitoring disk utilization or memory)?

- You can do both.
-
- Depends on what you are interested in.

Q10) What compiler optimizations are potentially harmful for joinpoints?

- Inlining, reordering of function calls, ...
- For dynamic weaving, basically everything that removes symbol information.

Q11) What about escape analysis, exceptions, dead code elimination?

- Yes, what about them?

Q12) Hidden cost for disengaged aspects?

- Dynamic weaving is going to come at a cost.

Q13) Are static/dynamic optimization and aspect?

- Some could be implemented as one, if they're crosscutting, e.g., caching.

Q14) Define the following terms (can they be defined?): flexible, extensible, adaptable, adaptive, configurable, static, dynamic?

- This goes into the CFP for the next workshop.

2. Type Safety in Operating Systems

Q1) How much do we really gain from type safety in operating systems?

- Aren't the actual problems beyond/outside what we(you) can achieve/address with type safety?

Q2) Why use a language designed for applications for OS development when it is obviously not made for that?

-

Q3) Isn't the money spent on not allowing people to shoot themselves in the foot better spent on educating them not to shoot themselves in the foot?

- Specifically: Isn't the money spent on figuring out how to use type-safe languages in OSs better spent on educating OS developers?

Q4) Killer question: How to convince the OS community and the industry that this is the way to go?

Q5) For running on the JVM, why not use fat pointers (array pointer + offset)?

- What to do with code that depends on its original encoding? How to make sure that you don't translate data?
- (e.g. what if you write bytes to a vector and then invoke a pointer in the vector).

Q6) Implementation benefits?

- Static type-checking helps catching implementation errors at compile time.

Q7) Runtime benefits?

- Substitutes address space separation/MMU.
- Fast communication between applications.
- Avoid runtime checks through static typing (less checking for system calls).

Q8) Why use application languages for OS development?

- To build an OS a language only needs few domain specific features (machine access operations).
- An OS is a very complex application and needs similar high level abstractions.
- Problems found in OS are similar to those found in applications like Database Servers, Web Servers, Transaction Systems. Why use different languages?

Q9) What is a Java OS?

- JavaOS is an OS that is written for the JVM machine model. It can be written in Java (Source Language).

Q10) Why spend money on type-safe OS methodology instead of educating OS developers?

- Even good programmers have a bad day and make mistakes.
- Debugging is very expensive and time intensive.
- Tools and Methodology scale better than (educated) developers.
- Good guys with good tools is the best combination.
- Common errors are eliminated. Writing OS still requires educated developers.

Q11) How to convince OS community/industry to use type-safe languages?

- Start with domains where problems addressed by type-safe languages are particularly grave.
- Type-safe languages allow to use cheaper programmers (save money).

Q12) Why not use fat pointers when running native code on the JVM?

- Fat pointers are feasible when compiling an unsafe source code language like C to JVM.
- They don't really help much when executing native code.
- Many applications are available as native code only.

3. Domain Specific Languages

Q1) What is the real use of AOP in an OS?

Same goals as in applicat

Q1) Why not strip away features from existing GPL that hinder analysis (for example require bounded loops)?

- Good way to design a syntax
- Might want to express some things that are not natural to express in a GPL

Q2) How do you combine DSLs. What happens if different DSLs interact.

Q3) How different are DSLs for different domains.

- Very: declarative vs imperative

Q4) Is it possible to have a common platform for DSLs (like .NET for GPL).

- Maybe for a family of families (eg OS resource management)

Q5) What to do when implementing a new member of the family and a DSL feature is missing?

- Complain to the DSL author, or maybe what you want to express is not appropriate

Q6) How to find the right granularity of program families for a DSL?

- Expertise in the domain, and perhaps in DSL design

Q7) Are Emacs LISP, SQL, Fortran, Cobol, shell languages, yacc, DSLs? Why not? Why?

- No, they are at best scripting languages

Q8) Is there a way to prevent reimplementing the same language features over and over?

- We need work for computer scientists

Q9) Can a DSL be turing complete?

- No, Turing complete = GPL

Q10) Why do many DSLs end up becoming Turing complete after a while (SQL, tex, latex, postscript, HTML) ?

- Bad influence from industry

Q11) Why use DSL and not code analysis?

- Some things are impossible to analyze in a GPL (termination)
- Analysis of DSL code only has to consider specific program patterns, that could be implemented in many different ways with a GPL

Q12) Is there a DSL to develop DSLs?

- Perhaps.

Q13) Can DSLs replace Type-safe Operating Systems and Aspect-oriented Programming?

- Yes.

Q14) Do we need a new domain specific language for every domain?

- Why can't we find a language that does for everything (and is extensible if need be)?
 - o Verification
 - o Or other kinds of domain-specific reasoning about a language (hacking)

Q15) Real world domains have the tendency to change. How do you find a language for a changing domain?

- Methodologies for designing DSLs to make extending the language easy
- A well-defined domain will not change so often. Try to isolate fundamental points of the domain, not implementation details.
- In a well-designed language, language abstractions may mirror future evolutions

Q16) When we have DSLed everything, how do all these DSLs integrate?

- In the context of the OS, there is a DSL per service
- Interaction and integration solved by service composition

Q17) Is it always implicit that a DSL is also about verification as well?

- Yes (3)
- No: want to make the life of the programmer easier
- Verification is the difference between a DSL and a scripting language.