Top12
Open-Source Software in Dependable Systems

Thursday 26 August

Organizers: Philippe David (ESA, The Netherlands), and Hélène Waeselynck (LAAS-CNRS, France)
Contact: Helene.Waeselynck@laas.fr

While Open-Source Software (OSS) is penetrating the software business at large, software-intensive applications having high dependability requirements have been little concerned so far. At a first glance, introducing OSS into such critical systems seems risky. The constraints imposed by adherence to certification standards may be deemed irreconcilable with the open-source development model. Still, the question is being taken seriously in domains like transportation, space, or nuclear energy: gaining acceptance into dependable systems might well be the new challenge for some mature OSS products.

10h30 – 12h
Introductory talks
Chair: Hélène Waeselynck (LAAS-CNRS, France)

OSS in Critical Systems: Motivation and Challenges – Philippe David (ESA, The Netherlands), Hélène Waeselynck, Yves Crouzet (LAAS-CNRS, France)

Trusting Strangers – Carl Landwehr (U. of Maryland, USA)

An Interdisciplinary Perspective of Dependability in Open Source Software – Cristina Gacek (U. of Newcastle, UK)

13h30 – 15h
Insights from OSS suppliers
Chair: Philippe David (ESA, The Netherlands)

Is Academic Open-Source Software Dependable? – Shigeru Chiba (Tokyo Institute of Technology, Japan)

Open Source in Dependable Systems: Current and Future Business Models – Cyrille Comar, Franco Gasperoni (ACT Europe, France)

An Open-Source VHDL IP Library with Plug & Play Configuration – Jiri Gaisler (Gaisler Research, Sweden)

15h30 – 17h
Insights from OSS integrators and general discussion
Chair: Andrea Servida (European Commission, Belgium)

Linux: a Multi-Purpose Executive Support for Civil Avionics Applications? – Serge Goiffon, Pierre Gaufillet (Airbus, France)

A Journey towards an OSS-Aware Organization – Jean-Michel Tanneau (THALES Research & Technology, France)

General discussion and concluding remarks by Andrea Servida (EC, Belgium)

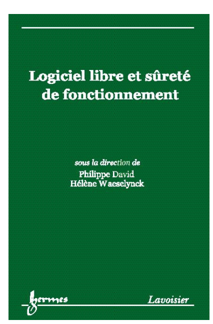
Open Source Software in Critical Systems: Motivations and Challenges

Philippe David, Hélène Waeselynck, Yves Crouzet
European Space Agency & LAAS-CNRS
WCC 2004-Top12

RIS
Network on Dependability Engineering

<http://www.ris.prd.fr>

- **Working Group on OSS and Dependability**
  (RIS members + participants from Industry ESA, SNCF, and Academia LSV, INRETS, IRISA)
- **Publication of a book**
  Hermes Science Publications
  (in French) -->



Some Facts

- **Some functions implemented by OSS are used in critical systems:**
  - Operating Systems
  - Communication protocols
  - language
- **OSS projects are more organised than one can usually think:**
  - Funded by associations or groups of industries that share a common interest.
  - OSS development is usually well organised.

➔ **Can we then expect benefits from using OSS in building critical systems?**

How we have conducted our analysis

- **Analysing the economical parameters for the use of OSS, in a global manner at system level.**
- **Exchanging information with industries and laboratories.**
- **Using feed back information from using COTS.**
- **Taking into account critical system requirements:**
  - Certification,
  - Detailed knowledge of the underlying technologies.
  - Maintenance aspects.

Evolution of critical systems

- **Critical applications are more and more widely used in our society.**
- **Production Cost is more and more constrained.**
  - ➔ Reuse is favoured instead of new software development: COTS.
- **Interoperability of systems is coming: systems of systems**
  - ➔ Use of interface standards is mandatory
  - ➔ Security must be taken into account.
- **Massive use of Software**
  - ➔ 48 kb onboard satellites in 1980 ➔ 1,2 Mb on Mars Express in 2003.
  - ➔ 25 Kb onboard A300B Airbus plane in 1974 ➔ 64 Mb on A380 in 2005.
- **Certification requirements extend to an increasingly larger set of industrial domains.**

Feed-back from using COTS in critical systems

- **System Integrator Needs**
  - Detailed knowledge of the COTS
  - COTS must adapt to the system
  - Certification file must exist
  - COTS must stay available during a 5-10 years period of time.
  - Long term Maintenance ensured for 10 to 20 years
  - Compliance with standards
  - Cost and details of the license must be negotiated
- **Encountered drawbacks**
  - Provider may not be interested in providing support
    - ➔ cost of support
  - Integrator does not and cannot know the details of the COTS
    - ➔ cost of the certification file
  - Diverging interests of user and provider due to the market evolution
    - ➔ cost of freezing the version
  - Freezing a version for a long time
    - ➔ maintenance cost
  - For small number of systems
    - ➔ cost of licenses is significant
  - Proprietary clauses may constitute a blocking point for system
    - ➔ negotiating the licenses is a key

Risk Mitigation

- **Risk due to COTS license**
  - Strategic problem.
  - Industry is used to manage it: license, property.
- **Risk due to COTS Failure**
  - Provider's liability is limited.
  - Failure propagation from COTS to the whole system is a real problem that must be managed by the integrator.
  - The system integrator has no/little knowledge about the COTS, support is necessary.
  - Confidence between provider and integrator is of prime importance. It is not sufficient when dealing with critical systems.
- **COTS provider disappearing**
  - Major industrial risk with no simple solution.
- **Risk due to OSS license**
  - Freedom of use.
  - GPL is contaminating other SW: Major point to be managed.
- **Risk due to OSS failure**
  - The system integrator is the only responsible.
  - Failure propagation must be managed by the integrator.
  - OSS source code is available, the integrator can acquire the technology. Support can be necessary.
  - The integrator must get confidence in the OSS. This is a major issue.
- **OSS evolution**
  - OSS can be maintained by the system manufacturer

Impact of the system maintenance

- **Life time of critical systems is quite long**
  - Satellites: 15 years
  - Command and control for nuclear propulsion in boats and submarines: 40 years
- **Maintenance issues are impacting the system design**
  - Architectural solutions must be used to minimize the impact of version updates.
  - ➔ The use of interface standards is favoured.
  - ➔ Wrapping mechanisms allow changing Software versions with minimum impact on the system.
- **Long Term Maintenance asks for risk mitigation actions to cope with the change of provider**
  - ➔ Availability of the source code is mandatory

Assets of using OSS in systems

COTS ➔ group of users ➔ standards for interface ➔ OSS

Specific, Proprietary ➔ Standard, Public ➔ Easier Interoperability

- No restriction to access the source code
- Does this access to source code help in easing the design/development/maintenance of critical systems?
- Several scenarios are encountered:
  1. Acquisition phase of the OSS Technology.
  2. Adaptation phase of the OSS to the system.
  3. Building the certification file.
  4. Operational maintenance.
  5. Putting in place a long term maintenance team.
  6. Managing major system evolutions.

Scenarios of use of the OSS

	Technology Acquisition	Adaptation to system	Certification documents	Operational Maintenance	Long Term Maintenance	Major Evolutions	Synthesis
Scenario 1 ✓No certification ✓No maintenance	Not necessary	Done by OSS Provider	Not necessary	Done by OSS Provider	Use of Source Code	Done by OSS Provider	No investment. Risk is low and accepted. *Situation in Space today*
Scenario 2 ✓No certification ✓Maintenance	Done through OSS Provider	Done by OSS Provider	Not necessary	Done by Integrator	Done by Integrator	Done by Integrator	Technology Acquisition. Investment in an in-house OSS maintenance team.
Scenario 3 ✓Certification ✓No Maintenance	Done through OSS Provider	Done by Integrator	Done by Integrator	Done by OSS Provider	Done by OSS Provider	Done by OSS Provider	Technology Acquisition. Certification by the Integrator. No maintenance on OSS.
Scenario 4 ✓Certification ✓Maintenance	Done through OSS Provider	Done by Integrator	Done by Integrator	Done by Integrator	Done by Integrator	Done by Integrator	Technology Acquisition. Certification by the Integrator. Investment in the OSS maintenance team.

Assets and drawbacks

- **Access to source code**
  - Allows mastering the evolutions of the software
  - Independence from any provider
  - Major risk: in case of failure, got source but without getting corresponding knowledge. This is the same with COTS.
- **OSS Technology Providers**
  - Same process as for COTS, without licensing problems.
  - Provided support is often of better quality than for COTS as the provider core competence is the OSS technology and not selling license.
- **Technology Acquisition**
  - Detailed Technology Acquisition on the OSS may cost several person.years
  - Investment is heavy on short and long term in order to maintain the OSS team during the project life time.

Dependability of the OSS

- **Some design infrastructure must be used to host the OSS as:**
  - The OSS is potentially a point of failure whose modes are not known.
  - The OSS functionalities may be too abundant or not fully suitable.
- **Use of wrappers**
- **Partitioning the critical system into different criticality levels.**
  - Error confinement mechanisms allow critical systems to be open for interoperability with other systems.
- **Security**
  - Should be taken into account as the OSS has been developed by a third party, often not known.

Certification

- Certification has a strong impact on the design of the system.
- Dependability and ability to be certified are not taken into account by OSS design.
  - ➔Reluctance of industry to use the OSS.
  - Must be performed by the industrial user.
- Our objective : to analyse the certification processes of the various industrial domains in order to
  - identify methods and efforts for allowing system certification when using OSS
  - ➔ OSS must demonstrate a competitive advantage for the system
  - ➔ Without introducing new risks

Certification: overview on various industrial domains

- Levels of criticality are ordered in a similar way in all the industrial domains.
  - DAL (Development Assurance Level) in aeronautics
  - SIL (Safety Integrity Level) in railway

Category	Railway	Aeronautics	Space	Nuclear
No Impact	SIL 0	E	/	/
Impact on system	SIL 1-2	C-D	critical	B and C
Impact on human lives	SIL 3-4	A-B	catastrophic	A

Impact of the safety levels on the system architecture: aeronautics

- **Safety analysis top down from system to all equipments that contribute to safety.**
- **Certification body has a dedicated referential for Software design and development, the DO-178B, who provides recommendations in the aim of guarantying the system safety:**
  - Electrical command of the planes are software implemented.
- **5 software categories (A to E) are defined**
  - Depending on the impact a software failure may have on the system.

System solutions to the use of software categories: aeronautics

Classification of failure conditions	Level of redundancy		
	0	1	2
Catastrophic	A	B	C
Dangerous	B	C	D
Major	C	D	D
Minor	D	D	D
No effect on safety	E	E	E
	DAL (Development Assurance Level)		

A critical software function able to lead to a catastrophic failure must be either:

- Not redounded. In this case, it is classified in software category A.
- Duplicated. Each of the two software versions is classified in category B.
- Triplicated. Each version is classified in category C.

Certification and dependability of OSS

- **A critical system can be designed from less critical functions only if they are redounded and the redundancies are managed according the safety requirements of the system.**
- **Communication protocol or operating systems are potential candidates for use at level C or D.**
  - ➔ Use of redundancies renders the certification feasible for use of OSS at level C or D.
- **Use of OSS at levels A or B implies a dedicated development process where the Software is specifically developed and certified accordingly.**
  - ➔ Building new OSS.

Development method

- **DO-178B defines objectives**
  - ◆ The certification case must contain proof elements that contribute to a negotiation between the industry and the certification body.
- **In nuclear, railway and space domains, a design and development method is imposed per category.**
- **Three major objectives:**
  1) Fault avoidance by applying rigorous development methods,
  2) Fault removal by using tests and integration tests,
  3) Protection from remaining faults through the use of dedicated functions for fault tolerance and robustness

➔ **It is possible to harmonise the certification processes among these domains: in terms of software life cycle and methods.**

Part of the certification effort can be shared by consortium of users

Bringing OSS to the level of use in a critical system requires two kinds of effort:

- **Generic tasks: depend only the software and results may be used by all system willing to embed the OSS.**
  - Documentation
  - Tests
- **Specific tasks: depend on the system, mainly oriented towards safety and hardware interface.**
  - Safety assurance plan. For SIL 1 et SIL 2 (C, D), safety requirements are quite limited.
  - Hardware integration.
    - Software test on hardware must be rerun for each project.

➔ **Development and test efforts are mainly system independent or must be re run anyway.**

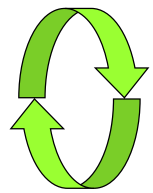
➔ **Certification effort can be anticipated.**

Usable methods for integrating an OSS in a critical system

- ***Analysing the certification standards of the various industrial domains for critical systems allows us to conclude that:***
  - A list of common method can be used to adapt and integrate an OSS in a critical system.
  - These methods depend on the criticality of the function and not on the industrial sector in subject.
- **Dedicated solutions exist to embed OSS in critical systems**
  - wrappers
  - Partitioning
  - Protection mechanism: security?
- **System architecture must be based on interface standards**
  - Favour the use of OSS components
  - Enhance the system life time and ease the maintenance

Expected benefits of using OSS: virtuous circle

- **Use of interface standards allows exchanging Software between projects and companies**
- **Building the OSS Certification file is a big effort, requires heavy investment but it can be shared.**
- **Reluctance of industry to use OSS comes from the perceived non compatibility of OSS with certification**
  - We demonstrated that this is not true when using proper architectural solutions at system level.
  - Risks are better managed than with COTS
  - Building the OSS Certification file must be started by a group of user companies.
    - Sharing the effort
    - Common use of the file that will be enriched from various operational practices.
- **Initiating the virtuous circle**



Conclusion (1/3)

- **Use of COTS has demonstrated some limitations.**
- **Use of OSS brings real assets:**
  - Comply to standards
  - Lower risk
  - Source code use for adapting and maintenance
- **Perceived drawback**
  - Non compatibility with the certification process but we demonstrated that solutions exist:

➔ **Architectural solutions to host OSS components**

➔ **Starting the virtuous circle by an industrial initiative**

➔ **Industry can now be beneficial in contributing to the OSS community**

Conclusion (2/3): initiatives for promoting the use of OSS in critical domains

- **Setting-up a industrially shared set of methods and tools to use OSS in our systems: repository on internet.**
- **Upgrading OSS to fulfil industrial constraints.**
  - Common methods and tools
  - Sharing the OSS
  - Tools are put at the disposal of users.
  - development environments used to produce critical systems
- **Evaluating OSS, and capitalizing on their use**
  - Validation : characterisation of failure modes and performance
  - Wrappers can be made available
  - Configuration of OSS for dedicated use or hardware
  - Starting the common activity towards a certification file

Conclusion (3/3): Future must be prepared

- **OSS is not the only open source item:**
  - Open hardware (VHDL or C models)
- **System engineering is more and more using simulation models from which code is automatically generated.**
  - Models must be made freely available
- **A new license**
  - Industrial needs are not consistent with GPL conditions. A lot of new ad-hoc licenses are emerging.
  - A license for industrial use of the OSS must be established.
  - Can be an European initiative

IFIP WCC Topical Day on
Open Source Software in Dependable Systems

Trusting Strangers
Open Source Software and Security

26 August 2004

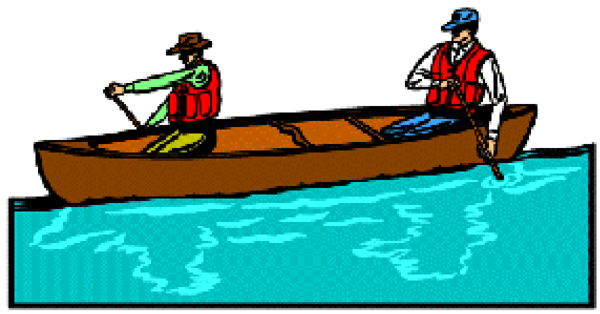
Presented by
Vipin Swarup
The MITRE Corporation

Carl Landwehr
Institute for Systems Research
University of Maryland

Outline

1. **Software and Trust**
2. **Certifying Security**
3. **Open vs. Closed**

Visible (inspect-able?) systems



Less visible

- Even a basic car like a Citroen 2-cv hides a lot under the hood
- Consider a modern airliner

What about this building?

Or this one?

(CDG terminal 5/23/04)

Or your microwave oven?

We rely on many anonymous strangers to design, build, deliver, and maintain critical systems

But it's not blind trust

- We have building codes and inspectors
- We have safety regulations
- We have product liability
- We have publicity when accidents and failures occur, and consumers react

Software is an unusual artifact

- Little physical substance, but can convey sensitive information and control significant energy
- Significant costs in design and implementation
- Low cost of replication
- Small changes to its representation can yield major behavioral changes to systems
- Usually licensed, rarely sold
- Licenses typically relieve producer from product liability

Certifying Software Systems

- Safety certification:
  - Baseline assumption: incompetence, not malice
  - Typically a combination
    - Development process controls
    - Inspection and testing
  - Additional strong economic factor:
    - consumer response to accidents
  - Status: not perfect, but reasonably effective

Certifying Security

- Baseline assumption: malicious attacker
- Common Criteria (CC) scheme
  - Permit separate specification of function and assurance requirements
  - Develop Security Target (specification)
  - Develop Target of Evaluation (implementation)
  - CC Testing Lab checks whether TOE meets ST
- Issues:
  - Unless relatively high assurance levels are requested, source code will not be reviewed by lab
    - And most flaws exploited in today's attacks are in the implementation, not the spec
  - Scheme remains component-oriented
    - Security is a system property
  - Cost-effectiveness unknown

Open vs. Closed

- Should we encourage/allow/disallow the use of open source software in security-critical applications?

  + Arbitrary tools can be used to investigate, modify, re-link, rebuild, analyze, the software
  + Third party can examine in as much detail as you can afford

  but

  - Liability for the results will rest with you
  - Lf you don't review the software, there's no guarantee anyone else has either
    - most of those "thousands of eyes" lack expertise and interest
    - some of them might be malicious

Is closed source better?

- Carries the producer's economic interest in the product – a potent factor
  + Can drive control of software development
  + For large companies, reputation is a factor
- But
  - Not much product liability for licensed software either
  - Hackers find flaws even without source access

Conclusions

- Caveat emptor
  - Neither open nor closed source produces "bullet-proof" software without specific investment for that purpose
  - Exposing source doesn't automatically improve its security properties
  - Neither does hiding it
- Seek product and architectural assurance
  - Process assurance is uncertain in a world of outsourced component software modules
- Exploit what you know, and what know you don't know
  - If you use open source, consider whether to reconfigure or rebuild
  - If you purchase closed source, investigate the developer's processes, motives, independent evaluations
  - Build system architecture taking these into account

Thank you!

Discussion?

Acknowledgements:
Thanks to Michael Hicks, U. Md for discussions
Thanks to Vipin Swarup, MITRE for presentation!

"I still don't have all the answers, but I'm beginning to ask the right questions."

An Interdisciplinary Perspective of Dependability in Open Source Software

Dr. Cristina Gacek

School of Computing Science
University of Newcastle upon Tyne – UK

Overview

- Context
- What is OSS?
- Preliminary Conclusions
- Evaluating the Dependability of OSS Products
- Deriving Dependability Insights from OSS Products
- Future Work

Context

- Within the DIRC (Interdisciplinary Research Collaboration in Dependability) project
  - 1 year activity
  - Feasibility study for further activities in the area of development of dependable systems using open source approaches
- Several students' dissertations
  - Investigating Open Source projects

What is Open Source Software (OSS)?

- Lack of precise use of the term
- Usually a combination of one or more of
  - Licensing model
  - Visibility of source code
  - Right to modify
  - Multiple reviewers
  - Multiple contributors

What is OSS?

- Open Source Definition (OSD)
  - Provided by Open Source Initiative (OSI)
  - Addresses legal and (some) economic issues
    - Ability to distribute software freely
    - Source code's availability
    - Right to create derived works through modification
- The many meanings of Open Source
  - View from various disciplines: CS, Management, Psychology, Sociology
  - Finding common and varying characteristics of open source projects

What is OSS?

COMMON

Adherence to OSD

Developers are users

- Modularity
- Visibility of software architecture
- Documentation and testing
- Accepting submissions
- Tool and operational support

VARIABLE

Starting point

Motivation

Community

- Balance of centralization and decentralization
- Meritocratic culture
- Geographical distribution

Software devel

Licensing

Size

- Community
- Code base

- Choice of work area
- Decision making process
- Submission information dissemination

OSS vs. "Traditional" Software

COMMON

Developers are users

Motivation

- Modularity
- Visibility of software architecture
- Documentation and testing
- Accepting submissions
- Tool and operational support

VARIABLE

Starting point

- Balance of centralization and decentralization
- Meritocratic culture
- Geographical distribution

Community

Software development support

Licensing

Size

- Choice of work area
- Decision making process

Preliminary Conclusions

- The term "Open Source" is often used in a vague manner
- OSS characteristics facilitate a better understanding
- As much variation exists between OSS projects as between any set of projects
- It is not meaningful to bundle all OSS products and projects into one category
  - Apache and Linux
  - Topologilinux and Frozen Bubble
  - 329 compilers in Freshmeat.net on 24/08/04

Stereotypes About the Dependability of OSS Products/Projects

- OSS products contain fewer faults because they have been reviewed by many people.
- OSS products are more secure because they have been reviewed by many people.
- OSS products have little to no design documentation available.
- Having little design documentation available does not impact an OSS project as negatively as it would a "traditional" one. The reason being that OSS developers contribute towards development for their joy and pleasure, and consequently are less likely to leave the project than an employee to change jobs.
- OSS products are developed by hackers in their free time, who only submit code for consideration once a high standard of quality has been achieved.

Evaluating the Dependability of OSS Products

- Like that of "traditionally" developed software
  - Needs to be done on a case by case basis
  - Different versions and releases of the same product must be considered individually
- Who would be responsible for pursuing certification?
  - One possible model: have interested companies work towards needed certification

Deriving Dependability Insights from OSS Products/Projects

- OSS characteristics are not restricted to OSS, hence insights from OSS can be used in other settings
- Studies are much easier to conduct in OSS than in "traditional" settings
  - Information available electronically
  - Time consuming to locate and collate related info
  - Key players usually receptive to queries
- Our results to date show a strong correlation between the quality of installation documentation and code readability

Future Work

- Study openness characteristics that foster more dependable systems
  - Which combinations of characteristics are beneficial?
  - Which combinations of characteristics are detrimental?
- Replicate results from OSS into "traditional" environments
- Explore avenues for adopting OSS into critical systems' settings

Is Academic Open-source Software Dependable?

Shigeru Chiba
Tokyo Institute of Technology, Japan

Topical Days: Open-Source Software in Dependable Systems
18th IFIP World Computer Congress
In Toulouse, France on Thursday 26 August 2004

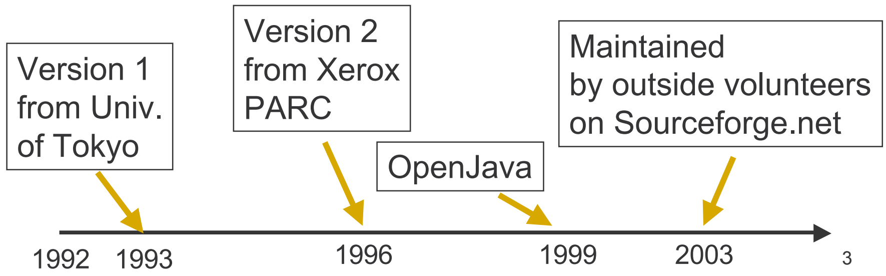
My experiences..

- My job
  - Research and teaching as a university prof.
- Research projects
  - Explore academic ideas
  - Develop software products as “proof of concepts”
  - Resulting products are distributed as Open Source Software.

2

OpenC++ and OpenJava

- Extensible C++/Java compiler
  - Method-call behavior, syntax, …
  - Research on reflective computing
- Project started in 1992



3

OpenC++ Users

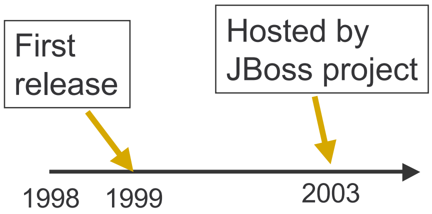
- Mainly used as a research platform
  - e.g.
    - To implement their middleware for dependable computing
      by J. C. Fabre at LAAS-CNRS
  - Downloads
    - >= 6,000 copies from 2000 to 2003
      both academia and industry

4

JBoss is the most downloaded

Javassist

- Java bytecode engineering library
  - Easy to use due to source-level abstraction
  - Research on Aspect-Oriented Programming
- Project started in 1998



JBoss is no.1 open-source J2EE server.
Market share:
No.1 WebSphere
No.2 WebLogic
No.3 JBoss 27%

Javassist Users

- Used as a library for implementing middleware for Web applications.
  - e.g. JBoss J2EE server, Tapestry, ...
  - Industry-strength open source software as well as research systems
- Downloads
  - >= 500/month in 2003Q4
  - >= 7,000/month as part of JBoss

The topic of this talk

Is Academic Open-Source Software (OSS) Dependable?

7

Yes!

- Code quality
  Industry-strength as other open source software
  - A larger user base tests on more platforms and finds more bugs.

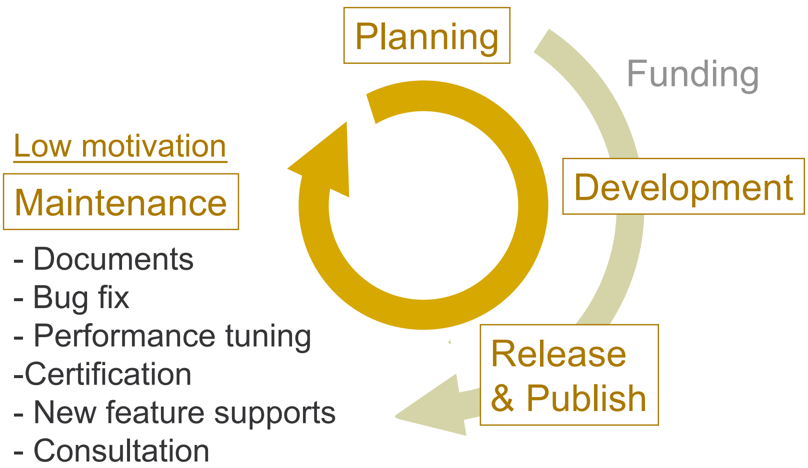
8

Yes!? (I'm not sure…)

- Long-term supports and maintenance
  - Key to dependable software
- Academic open source project
  - The goal is to publish academic papers.
  - It is funded; not a volunteers' project.
  - After the project ends, the software is…

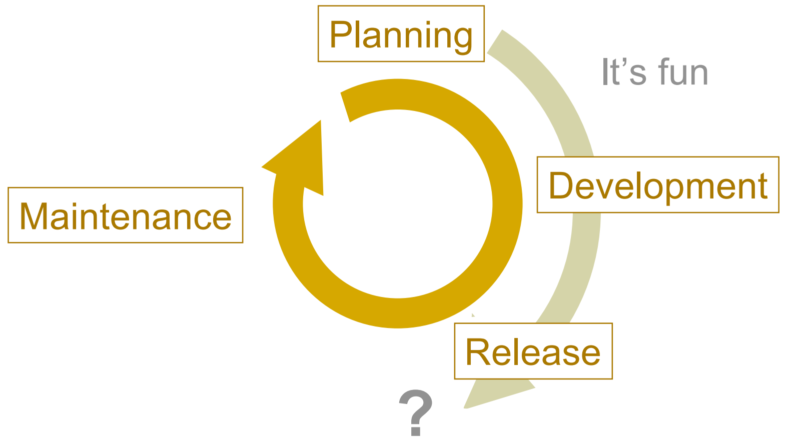
9

Life Cycle of Academic OSS



10

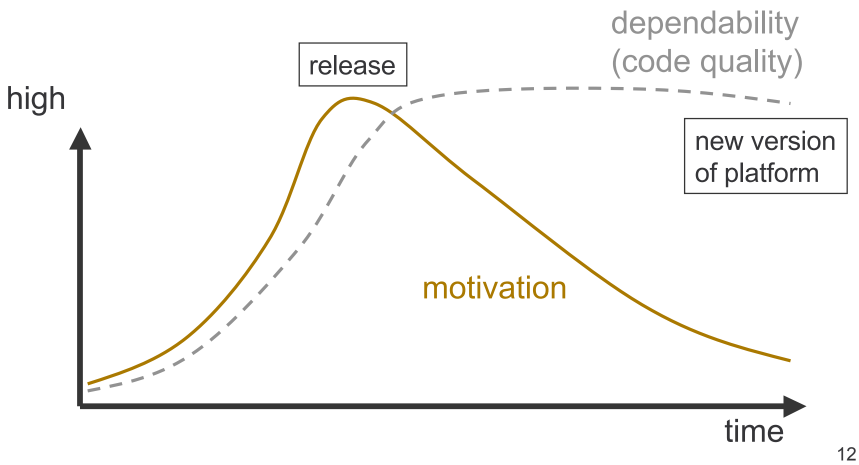
Life Cycle of Regular OSS



11

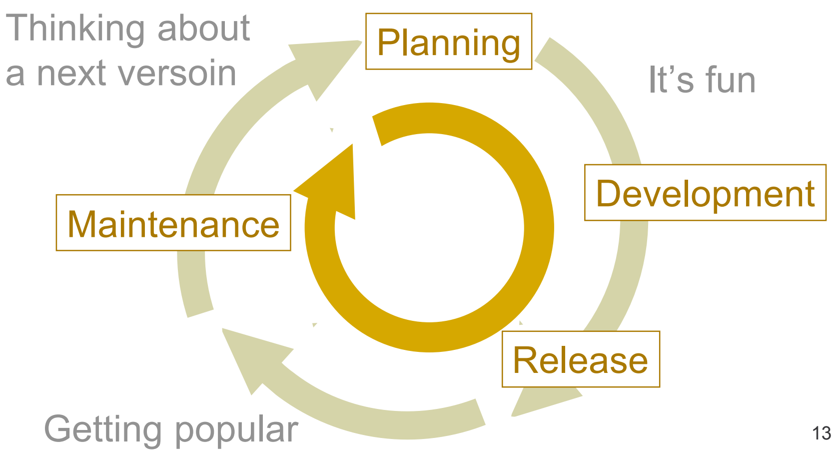
Software development is fun.

- But…



12

Life Cycle of Successful OSS



13

Short Release Cycle

- Motivates developers
  - Thinking a new version is fun.
- Bad for dependability
  - Which release fixes this bug?
  - Which release is stable?

  - Not perfect compatibility between releases.

14

Professional Open Source

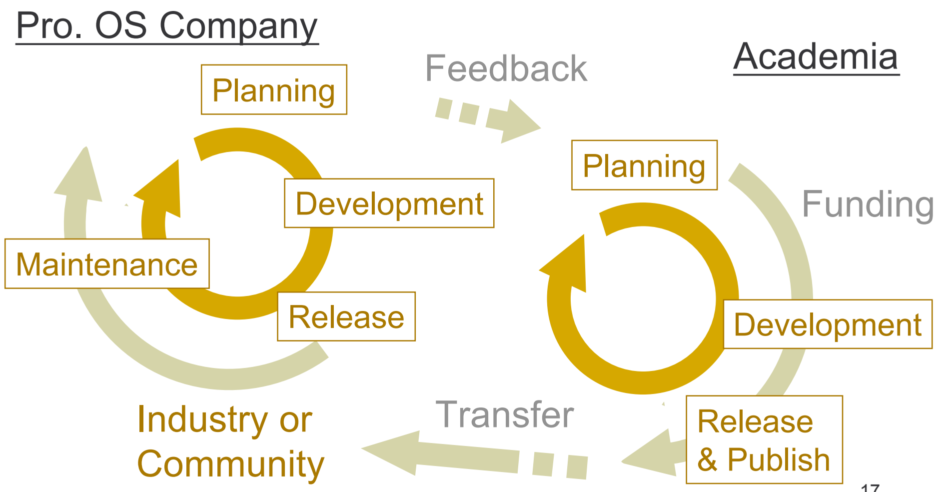
- Business model of JBoss Inc.
  - Employees spend their time
    - 50% for new OSS development
    - 50% for maintenance of their old OSS
  - The company sells supports and consultation of old OSS by the authors.

Open Source Software is not free!

To Make Academic OSS Dependable

How can we keep motivation?

Technology Transfer?



Who is interested in academic OSS?

17

Final Remarks

- If you use academic OSS
  for a commercial dependable system,
  - Acquire the OSS project,
    - And start Professional OSS business
  - Contract with the OSS developer for consultation, or
  - Hire the OSS developer if she is a student.

18

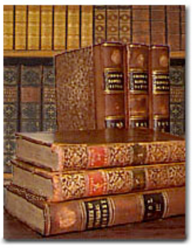
Free Software - Open Source Business Models

Franco Gasperoni
gasperoni@act-europe.com

Legal Background: Copyright

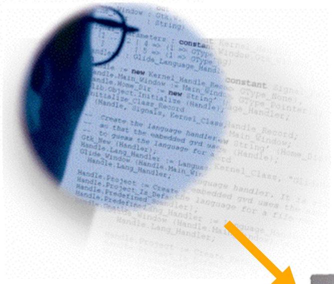
Exclusive legal right of copyright holder to

- Copy
- Distribute
- Modify
- Display
- Perform
- Exploit a work



Software and Copyright

- Both source and object code can be copyrighted



- Loading software onto a computer is considered copying

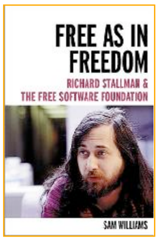
Copyright & Licensing

- A copyrighted work cannot be copied unless…
- … the copyright holder grants you a license …
- … permitting copies under specified circumstances

Virtually all software today is sold with a license

The Free Software Movement (ca 1980)

- *Freedom* to run
- *Freedom* to redistribute copies
- *Freedom* to study and adapt
- *Freedom* to improve it and release the improvements



Free Software Is Copyrighted Software

To protect these freedoms *Free Software* comes with a license

- The General Public License (GPL)

From this point there is NO DIFFERENCE between

GPL: The License of *Free Software* (FS)

- **The GPL is written to favor FS users**
- **Specifically the GPL guarantees:**
  - *Freedom* to run
  - *Freedom* to redistribute copies
  - *Freedom* to study and adapt
  - *Freedom* to improve it and release the improvements
- **Examples of FS:**
  - *Emacs, GCC, GNAT Ada, GNU/Linux, …*

The Meaning of *Free* in FS

Freedom

- **You can sell it**
- **You can make it available for free**

FS is a matter of liberty not price

Open Source Software (OSS)

- **Providing the sources** *(under some license)*

+

- **Encouraging a wide community to participate in development**

There have been abuses in the licenses used

- **Open Source Initiative (OSI) created to**
  - Define what licenses qualify as “Open Source”

The OSS Movement

- **Attractive to major companies (e.g. IBM, SGI, HP, …)**
  - Can leverage on a larger developer community
- **Claims are made for better quality, better security etc.**
- **In practice:**
  - Some OSS projects work, some don’t.
  - Some OSS software is high quality, some is not.
  - Some projects make sense as OSS some don’t.

FS and OSS

- **One of the important freedoms for FS is**
  - The freedom to modify, which means that sources are available.
- **So it is often, but not always, the case that FS:**
  - Ends up with an open source community participating in development.
  - E.g. Linux
- **Not all OSS projects are FS because of the license**

FS/OSS and COTS

Commercial Off-The-Shelf (COTS)

- **Most people look for COTS software**
  - Economies of scale
  - Reduced Costs
  - Inexpensive way to stay with the state of the art in technology
  - User community

COTS and Closed-Source Software

- **Two big downsides**
- **Vendor lock in for support**
  - Only the vendor can provide support
  - This can be locked in with licenses etc
  - If the vendor goes bankrupt, too bad
  - Source escrows are not much help
- **Vendor lock in for modifications**
  - If the software does almost what you want, but not quite, you have to ask the vendor for changes
  - This can be arbitrarily expensive

Free Software Licensed COTS

- Fixing the two big downsides of COTS
- NO vendor lock in for support
  - Everyone has access to the sources
  - Anyone can provide support
  - You can even build your own support
  - If there is a demand other companies will compete
- NO vendor lock in for modifications
  - Everyone has access to the sources
  - Anyone can do modifications
  - You can do modifications yourself if you like

COTS + FS = COTS without the risks

Worrying about Licenses and Quality

FS, OSS, and Proprietary Software share 3 common truths

- CHECK THE LICENSE
  - Make sure it is suitable for your use
- CHECK THE QUALITY
  - No software license guarantees quality
  - Use your normal procedures to ensure that you choose quality software
  - Buy SW products whose business model aligns with your quality needs

"Quality" and FS/OSS Business Models

FS/OSS and Dependable Systems

SW in a dependable system:

- Part of an auditable & repeatable process
- With stringent "quality" requirements
- What are the quality guarantees for FS/OSS ?

FS/OSS Product With No Support

- Supplier sells FS/OSS applications
  - Perhaps with some installation help
- E.g. previously commercial GNU/Linux distributions
- Can check quality by inspecting the sources … ☺ ☹
- This is an advantage over conventional proprietary SW
- Not particularly attractive to developers of Dep. Sys.

Dual License

- Available to FS/OSS companies that own 100% copyright
- Whose products are included in the sw developer's code
- E.g MySQL, Cygwin
- Relies on vendor lock in
- No additional advantage over previous model

Infrastructure Provider

- **OSS development website**
  - E.g. OSDN, SourceForge
- **Leverages on large developers community**
- **Free for basic services, fee for advanced web browsing**
- **Revenue from some advertising**
- **SourceForge Enterprise Edition**
  - To manage and execute offshore and distributed team development
- **Interesting for large/distributed teams**

Pure Service

- **E.g. Alcove, IBM Global Services for GNU/Linux**

Different from "traditional" service models in that:

- **Consultants have access to the sources**
- **Can contribute to OSS efforts**
- **Allows deeper level of consultants know-how**

Sell the Artifacts

- **SW in a dependable system:**
  - Part of an auditable & repeatable process
  - With stringent quality requirements
- **SW in a dep. sys. = sources + build scripts + artifacts**
- **Provide the artifacts for FS/OSS product and sell them**
  - Creation of artifacts is not the main focus of FS/OSS

Software Coops

- **Coop to share resources and know how**
- **To develop artifacts for FS/OSS application**
- **More generally to guarantee FS/OSS quality**
- **For the members of the coop**

Leveraged Service

- **FS/OSS product with expertise-based service**
- **Provided by the developers of the FS/OSS product**
  - E.g. AdaCore and GNAT Pro
- **Quality guaranteed by aligning interests with customer's**
  - Subscription-based model
  - Quality can be verified on an ongoing basis
  - Quality feedback loop in place
  - If poor quality/service subscription not renewed

Quality is an ongoing process

Conclusion

Common truths of FS, OSS, and Proprietary Software:

- **CHECK THE LICENSE**
- **CHECK THE QUALITY**
- **CHECK THE BUSINESS MODEL**
  - Make sure it is aligned with your interests

GRLIB Open-Source VHDL IP Library

Jiri Gaisler

Gaisler Research

jiri@gaisler.com

Introduction

- High device density (ASIC & FPGA) has led to a larger number of new SOC designs
- An improved design methodolgy is needed to allow cost-efficient development of complex SOC systems
- For this pupose, Gaisler Research has developed a open-source VHDL IP library for both commercial and space-based applications.
- This presentation will describe the concept of the IP library and provide details on some of its cores, including the LEON3 SPARC processor.

Common SOC design problems

- Merging of 3-party IP cores may cause several problems:
  - Harmonisation of interfaces (on-chip buses, irq ...)
  - Merging of synthesis and simulation scripts
  - Mapping of technology specific cells (RAM, pads)
  - Name space conflicts, CAD tool specific syntax
  - Licensing issues
- Problems for space applications
  - SEU hardening
  - Portability and long-term support

GRLIB design goals

- Efficient and unified SOC design IP library with:
  - Common interfaces
  - Unified synthesis and simulation scripts
  - Target technology independent
  - IP-vendor independent
  - CAD tool independent
  - Open and extensible format
  - (SEU tolerance for space applications)

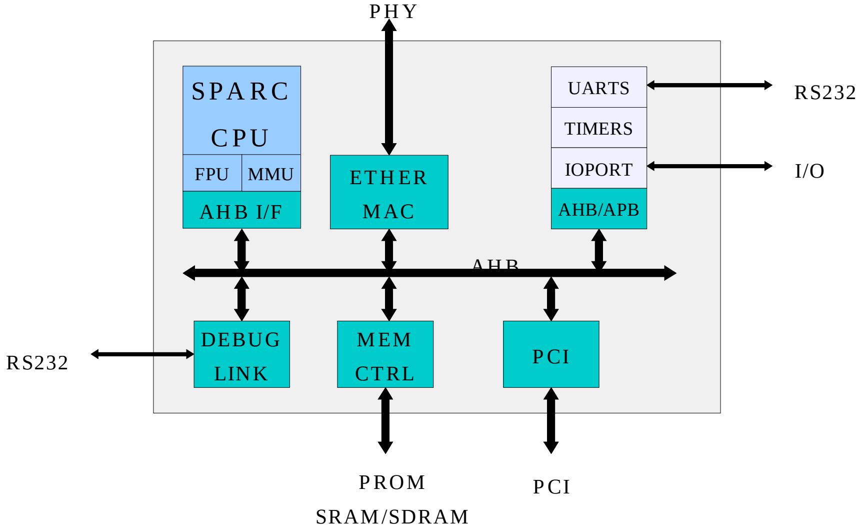
GRLIB implementation overview

- Based around AMBA-2.0 on-chip bus (ARM)
- PCI-style plug&play support for AMBA configuration:
  - Device & vendor identification
  - Address and interrupt configuration
- Vendors and cores isolated through use of VHDL libraries
- Portability achieved through RAM and pad wrappers
- Automatic generation of synthesis and simulation scripts
- Supported tools: Mentor, Cadence, Synopsys, Synplify, ISE
- New cores, CAD tool scripts or tech wrappers easily added

GRLIB IP Cores

- 32-bit LEON3 SPARC processor
- GRFPU IEEE-754 floating-point unit
- 32-bit PCI bridge with FIFO and DMA, PCI trace buffer
- Ethernet 10/100 Mbit Ethernet Controller
- PC133 32-bit SDRAM controller
- 32-bit PROM/SRAM controller
- AHB controller and APB bridge with plug&play support
- Utility cores: uart, timer, interrupt control, GPIO, ...
- Memory and pad wrappers for FPGAs and ASIC

Sample GRLIB SOC design



AMBA plug&play support

- Inspired by PCI plug&play method
- Distributed address decoding
  - => AHB/APB cores can be inserted/removed without modifications to arbiter or address decoder/multiplexer (!)
- Configuration table automatically built and readable from bus
- Address mapping and interrupt assignment through generics
- Pure VHDL implementation, no external SOC tools needed

SOC design VHDL code

```
ahb0 : ahbctrl        -- AHB arbiter/multiplexer
port map (rstn, clkm, ahbmi, ahbmo, ahbsi, ahbso);

u0 : leon3s                 -- LEON3 processor
generic map (ahbndx => i, fabtech => FABTECH, memtech => MEMTECH, isetsize => 1, dsetsize => 1)
  port map (clkm, rstn, ahbmi, ahbmo(i), ahbsi, leon3i(i), leon3o(i));

sd0 : mctrl                 -- PROM/SRAM/SDRAM memory controller
generic map (ahbndx => 0, apbndx => 0, apbaddr => 0, sden => 1)
port map (rstn, clkm, memi, memo, ahbsi, ahbso(0), apbi, apbo(0), wpo, sdo);
end generate;

apb0 : apbmst                     -- AHB/APB bridge
generic map (ahbndx => 1, memaddr => 16#800#)
port map (rstn, clkm, ahbsi, ahbso(1), apbi, apbo );

uart0 : apbuart             -- UART 1
generic map (apbndx => 1, apbaddr => 1,  irq => 2)
port map (rstn, clkm, apbi, apbo(1), u1i, u1o);

irqctrl0 : apbictrl               -- interrupt controller
generic map (apbndx => 2, apbaddr => 2, ncpu => NCPU)
port map (rstn, clkm, apbi, apbo(2), irqi, irqo);

timer0 : gptimer                  -- timer unit
generic map (apbndx => 3, apbaddr => 3, irq => 8)
port map (rstn, clkm, apbi, apbo(3), gpti, open);

pci0 : pci_target generic map (ahbndx => 1, device_id => 16#0210#, vendor_id => 16#16E3#)
port map (rstn, clkm, pciclk, pcii, pcio, ahbmi, ahbmo(1));

eth0 : eth_oc
generic map (mstndx => 2, slvndx => 5, ioaddr => 16#B00#, irq => 12)
port map ( rst => rstn, clk => clkm, ahbsi => ahbsi, ahbso => ahbso(5),
ahbmi => ahbmi, ahbmo => ahbmo(NCPU+dbg+pci), ethi => ethi, etho => etho);
```


SOC design simulation

```
VSIM 1> run
# LEON3 Demonstration design
# GRLIB Version 1.0
# Target technology: infered,  memory library: infered
# ahbctrl: AHB arbiter/multiplexer rev 1
# ahbctrl: Common I/O area at 0xfff00000, 1 Mbyte
# ahbctrl: Configuration area at 0xfffff000, 4 kbyte
# ahbctrl: mst0: Gaisler Research        Leon3 SPARC V8 Processor
# ahbctrl: slv0: Gaisler Research        PROM/SRAM/SDRAM Controller
# ahbctrl:       memory at 0x00000000, size 16 Mbyte, cacheable, prefetch
# ahbctrl:       memory at 0x40000000, size 16 Mbyte, cacheable, prefetch
# ahbctrl: slv1: Gaisler Research        AHB/APB Bridge
# ahbctrl:       memory at 0x80000000, size 16 Mbyte
# apbmst: APB Bridge at 0x80000000 rev 1
# apbmst: slv1: Gaisler Research        Generic UART
# apbmst:       I/O ports at 0x80000100, size 256 byte
# apbmst: slv2: Gaisler Research        Multi-processor Interrupt Ctrl.
# apbmst:       I/O ports at 0x80000200, size 256 byte
# apbmst: slv3: Gaisler Research        Modular Timer Unit
# apbmst:       I/O ports at 0x80000300, size 256 byte
# apbmst: slv7: Gaisler Research        AHB Debug UART
# apbmst:       I/O ports at 0x80000700, size 256 byte
# eth_oc5: Opencores 10/100 Mbit ethernet MAC, rev 0, irq 12
# gptimer3: GR Timer Unit rev 1, 16-bit scaler, 2 32-bit timers, irq 8
# apbictrl: Multi-processor Interrupt Controller rev 1, #cpu 2
# apbuart1: Generic UART rev 1, irq 3
# ahbuart7: AHB Debug UART rev 0
# leon3_0: LEON3 SPARC V8 processor rev 0
# leon3_0: icache 1*2 kbyte, dcache 1*1 kbyte

# cpu0: 0x00000000    flush  0x0000
# cpu0: 0x00000004    sethi  %hi(0x00001000), %g1  [0x00001000]
# cpu0: 0x00000008    or   %g1, 0x00c0, %g1  [0x000010c0]
# cpu0: 0x0000000c    mov  %g1, %psr
# cpu0: 0x00000010    mov  0, %wim
# cpu0: 0x00000014    mov  0, %tbr
# cpu0: 0x00000018    mov  0, %y
```

LEON3 SPARC V8 Processor

- 7-stage pipeline, multi-processor support
- On-chip debug support unit with instrcution trace buffer
- 250/400 MHz on 0.18/0.13 um, 250/400 MIPS
- Virtex2pro: 125 MHz, Actel RTAX: 33 MHz
- SEU tolerance by design for space applications
- All on-chip ram protected against SEU:
  - 136x32 bit register file: 4-bit parity and duplication
  - Cache rams use 4-bit parity and forced miss on error
  - No timing penalty, < 0.5% area overhead (on RTAX)

LEON3 Advanced floating-point unit

- High-performance single/double precision FPU (GRFPU)
  - IEEE-754, fully pipelined, 4 clock latency
  - ADD/SUB/MUL/DIV/SQRT/COMP/CONV
  - Dual execution units, parallel processor interface
  - Fault-tolerance against SEU effects
- 150/250 MHz, 150/250 MFLOPS on 0.18/0.13 um, 100 Kgates
- 40 MHz on Virtex-II, 9,000 LUTs
- Too large to fit on RTAX devices
- Can be used for DSP designs (custom or processor-based)

GAISLER RESEARCH

GRLIB Master/Target PCI

- Implements PCI 2.1 standard (32-bit, 33 MHz)
- Configurable FIFO depth
- DMA channel for independent block transfers
- 45/75 MHz, 9% area of RTAX2000S
- Full SEU protection through 4-bit parity and duplication
- No timing penalty, 4 RAM blocks overhead on RTAX

Synthesis results

Core	Cells	% of RTAX2000	Mhz
LEON3 + caches	3650	15.00%	35
PCI, master/target + DMA	2750	9.00%	45/70
10/100 Mbit Ethernet MAC	2200	7.00%	65
PROM/SRAM controller	500	2.00%	75
SDRAM controller	550	2.00%	75
LEON3 SOC system with: PCI, memory ctrl, timers, uarts, Irq ctrl, GPIO, ethernet MAC	16250	51.00%	33

LEON3 multi-processor support

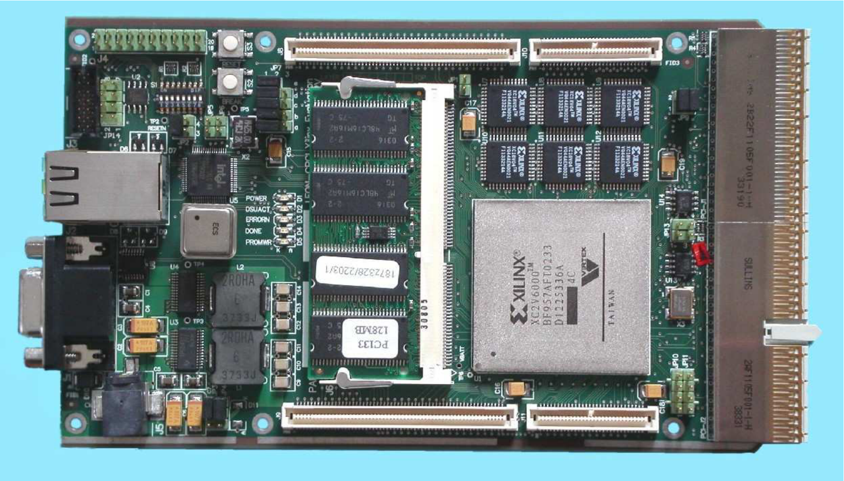
- LEON3 processor core + caches = 3 mm2 on 0.18 process
- Multi-processor system possible without area problems
- More than 4 cores not practical due to memory bandwidth
- Asymmetric configuration possible, e.g.2 'main' processors with FPU/MMU + 1 I/O (DMA/Interrupt) processor
- Multi-processor DSU and interrupt controller available
- 4-processor system fits on XC2V3000 FPGA @ 80 MHz
- 4-processor system fits on a RTAX2000 @ 25 MHz

GRLIB Support tools

- GRMON plug&play debug monitor
  - Debug 'drivers' for each specific IP core
  - Modules allow IP vendors to provide own drivers
- GRSIM modular simulator
  - Modular, re-entrant simulator based on TSIM
  - Can simulate any number of buses, cores or cpu's
  - Vendor independent models
  - Allows hardware/software co-simulation!

LEON3/SOC Development board

- Low-cost LEON CPCI FPGA development board available with **XC2V6000**, SDRAM, Flash, SRAM, 100-Mbit Ethernet



GAISLER RESEARCH

GRLIB availability

- Freely available in source code under GNU GPL
  - Valuable tool for academic research
  - Improves test-coverage due to large user-base
  - Allows early prototyping and try-before-buy
- Initial release September 2004
- Commercial licensing possible without restrictions
- The fault-tolerant version of the cores and the FPU are not initially released in open-source, but the long-term strategy is to release all cores under GPL.

GAISLER RESEARCH

Serge GOIFFON
Pierre GAUFILLET
AIRBUS France

Linux
A multi-purpose executive support for civil avionics applications ?

Civil avionics software context

- **Main characteristics**
  - Required dependability
  - More and more software intensive : From 23Kb to 100Mb and more
  - Synchronous and asynchronous architectures
  - Very long lifetime compared to hardware components
  - Integrated Modular Avionics concepts as new paradigm

- **Development process based on DO-178B/ED-12B**
  - Guidance for satisfying airworthiness requirements
  - Accepted by industrials
  - Define processes and processes data
  - Level of assurance and completion criteria depend on software criticality level

Highly critical avionics systems are not considered here !

The Operating System : a key component

- **O/S main objective**
  - Offers execution model and standard API to avionics applications
  - H/W access through drivers and generic services : no impact on applications if hardware changes
  - Portability, interoperability and reuse-ability of applications

- **Linux as a multi-purpose O/S candidate**
  - Open source based on common adopted standards -> portability, interoperability
  - Follows cooperative development model -> distributed knowledge, innovation
  - Adaptable, reliable, scalable, fits to a wide range of hardware components
  - Widely used and today mature for embedded market

- **3 steps for appropriation in avionics**
  - **Embedding Linux on avionics specific hardware platform**
  - **Host multi-level critical software : partitioning properties**
  - **Make Linux ready for DO-178B certification**

Industrial standards : POSIX vs ARINC 653

Some features comparison ...

POSIX	ARINC 653
Event driven execution model	Cyclic based execution model
Multi-processing, multi-threading execution model	Same as POSIX but terminology used is partition for process, and process for threads
Priority preemptive scheduling for processes and threads	Within partition time slice : priority preemptive scheduling of A653 processes with deadline management
No temporal partitioning	Temporal partitioning : fixed allocation of time slices to partitions in a repetitive time frame pattern
I/O interrupt driven	I/O polling and I/O partitioning
Memory segregation	Same as POSIX but terminology is spatial partitioning
Socket	Sampling and queuing ports on I/O
Inter Process Communication	Sampling and queuing ports on RAM
Mutex and condition variables	Buffer, blackboard, semaphore, event
Timers	No timers like POSIX but API service to wait for timeout
Signal management	Health monitoring

Linux for embedded and real-time systems

- Several OSS solutions based on Linux have been developed :

RTAI	KURT	QLinux
RTLinux	ADEOS	...
Linux/RK	RedLinux	

- The 2.6 kernel features low latency and preemptible kernel, and is now ready *out of the box* for soft real time systems.

- Today, some projects aiming to bring Linux to the required maturity for embedded uses are in progress :
  - Carrier grade Linux (telecoms) – high availability, hot swappability, kernel and driver robustness
  - FlightLinux (NASA) – Linux in Space systems
  - SELinux (NSA) – security enhanced Linux

Embedding Linux on an avionics platform

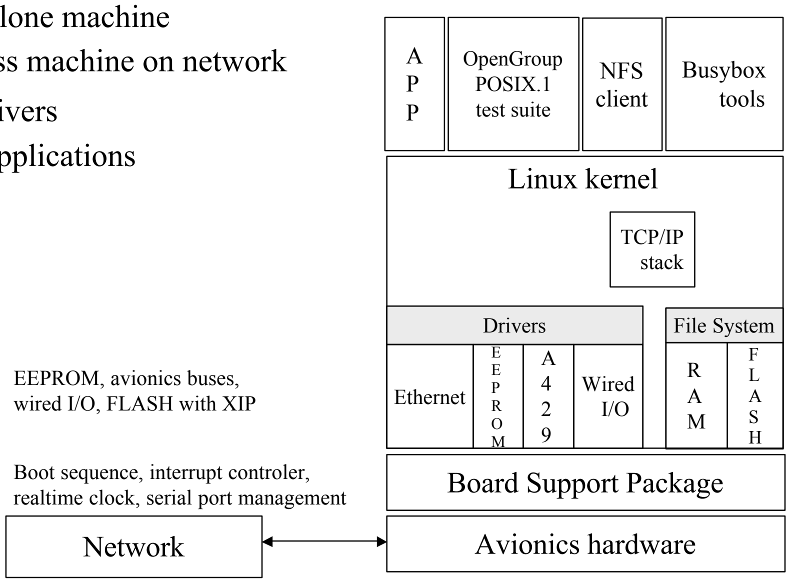
- **Research project**
  - Replace existing POSIX RTOS by Linux, in avionics platform, without changing existing applications

- **Targets**
  - Acquire kernel internals knowledge (drivers, memory management, file systems, scheduling, synchronization, time management, ...)
  - Verify the Linux API conformance to the replaced O/S (limit the effort of porting existing applications to the new Linux platform)

- **Results**
  - Linux integrated on a i486 avionics platform with network capabilities
  - Reduced kernel footprint (memory, drivers, file system, ...)
  - Reduced common Unix tools footprint (using Busybox)
  - Adapted avionics I/O drivers and FLASH PROM file system with eXecute In Place capability
  - Re-use of an existing Ethernet driver (fast prototyping)
  - Avionics applications successfully migrated to this new environment

Embedded Linux diagram

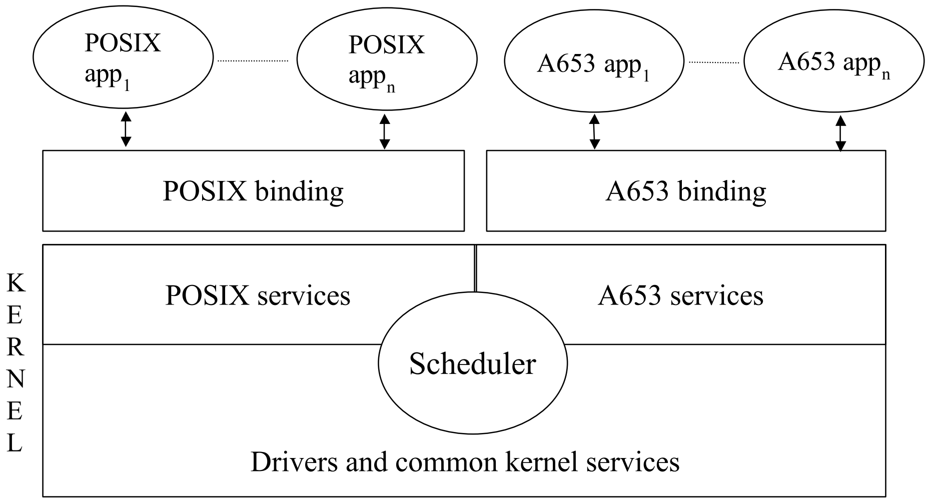
1 – standalone machine
2 – diskless machine on network
3 – full drivers
4 – with applications



Host multi-level critical software

- **Research project**
  - Host avionics applications, based on Integrated Modular Avionics concepts and ARINC 653 standard, on Linux platform
- **Targets**
  - Implement ARINC 653 cyclic scheduler and temporal partitioning
  - Implement a standalone ARINC 653 API (not relying on POSIX services)
- **Results**
  - ***Scheduler adapted to run both ARINC 653 and POSIX applications***
  - ***Sampling and Queuing ports attached to RAM, Unix sockets or AFDX (Avionics Full DupleX Ethernet) ports***
  - ***Static allocation of A653 system resources and dynamic control of their use***
  - ***Management of process deadlines***
  - ***Linux Trace Tool adapted to view ARINC 653 events (context switches of A653 processes, API calls, partition switch, ...)***

Host multi-level critical software diagram



Why Linux is not ready for DO-178B ?

- **From the DO-178B viewpoint**
  - No development and verification plans
  - Heterogeneous and complex development environment (distributed over Internet, multi-platform, etc.)
  - No universal requirement, design and code standards
  - No design document
- **But, from the product viewpoint**
  - OpenGroup testing environment provides test suites for POSIX conformance
  - Reliable software, modular architecture
  - Co-operative and hierarchically structured development model with centralised version management
  - Product reviewed and tested by peers

How to make Linux ready for DO-178B ?

- **Produce missing certification material using reverse engineering**
  - **Develop tools to extract semantic from code and produce kernel static and dynamic design**
  - **Focus on descriptions of the main kernel internal mechanisms**
- **Validation**
  - **Compliance to standard, robustness, kernel profiling and performance characterization**
- **Properties analysis**
  - Worst Case Execution Time, stack consumption, proof of properties in complex algorithms
- **Development**
  - Take part in the kernel development process to provide simple and deterministic algorithms in strategic parts of the software (memory management, scheduling, file system management)
  - Provide static allocation and dynamic control of system resources
  - Provide robust spatial and temporal partitioning

Conclusion

- **Studies show using Linux in an avionics environment is possible.**
- **The main problem for certification is the predominant part of the process objectives of the DO-178B compared to a product objective approach...**
- **Linux gives low cost access to reliable and adaptable technology but appropriation cost for dependable systems is not negligible.**
- **Industrials need to work in partnership with labs and Linux experts to share the cost of reverse engineering activities.**
- **Those certification activities should be processed in an Open Source project.**

AIRBUS

**AN EADS JOINT COMPANY
WITH BAE SYSTEMS**

OSS in the industry : the THALES example

Jean-Michel Tanneau
August 2004

jean-michel.tanneau@thalesgroup.com

Corporate Department

THALES

Introduction

The point of view : THALES
- ⇨ (Software dominant) Systems integrator

The context
- Increase of complexity & Price reduction
- Conflicting lifecycle : Technology - COTS Versus Systems
- Strong requirements : Reliable, Secure, Flexible, Configurable, Scalable, Available & Maintenable in LT
- Small volumes
- COTS era (Perry directive)

Objectives
- Increase performance (**effectiveness**): *quicker, better, cheaper*
- Improve durability of R&D investments (core business)

R&D software strategy : 2 of the priorities
- ¬Open architectures & Standardization
- ¬Sharing & cooperation on generic technologies (non core business)

Is OSS an opportunity to meet objectives and R&D strategy?

- How to benefit from the product ?
- How to benefit from the development process ?
- How to benefit from the mechanisms of « value creation » ?

OSS & Thales

Two phases

- Since 1999 : Usage of OSS (in business)
  - Main focus : To control risks
- Since 2002 : Use of OSS as a process
  - Main focus : To leverage opportunity

One approach, a mix of

- Strategic approach and (Technical) Change management

How to benefit from

OSS as products (technical objects)

Why to take an interest in OSS? COTS drawbacks

- Uncertainty
  - Product (black box) and delivered information (claimed Vs actual behaviour)
  - (product and editor) Strategy : evolution, roadmap, business model
  - Market (continuous restructuring)
- Subordination to a sole provider (monopoly)
- Divergent interests
  - “mass market” driven : progressive disinvolvement with our business
    - Certain domains considered as « niche » market
  - Shortening of COTS life cycle – Impact on Quality
- Others
  - COTS is intrusive (architecture / design)
  - Support
  - Cost ?

Use of OSS - Stakes

Opportunity

- Same advantages as COTS : productivity (time to market-cost to market) & added-value
- New source of provisioning (opens the market)
- Providers independence (durability of components) - **Control over system life cycle**
- A spreading (free & competitive) supply
- Trends : support from large IT companies & institutional users (administrations & MoD)
- Community based AND commercial **support** (free & competitive market)
- White box : secure (auditable), adaptable, predictable (certification)
- White process : evolution, quality (fast bug corrections)
- Users & technology driven
- **Standards based (Interoperability) - Commodification**
- !!! TCO ?

Threats / Brakes

- IPR (!!! **OSS licenses**)
- Warranty and Liability
- Software patents - LZW (GIF), MP3, SCO Vs. IBM lawsuit
- Diffuse and **unequal (quality) supply** - (!!! care to not generalize: OSS is not a “guarantee of quality”)
- Still an **external component**
- **Continuous evolution**
- Mixing many OSS
- Complexity (skills/training required)
- Un-grasped world + FUD

Legal issues

License analysis

- Is it an OSS ? (OSD compliant)
- Existence of third party patents ?
- Identification of restrictions / conditions related to redistribution (with or without modification)
- If clarification needed, apply to the author

Usage in a specific programme

- Usage is analyzed & documented (software architecture)
- Compliance with contractual requirement and regulation
- If many OSS used, check that their licenses are compatible

The Component Evaluation / Selection process

Aims

- reduce and/or delay risk occurrence - mitigate impact
- Effectiveness : fulfills technical requirements
- Confidence (now & mid-term)
- Economic efficiency : TCO, know-how capitalization (ROI)

⇨ To get *"The right product, at the right time, at the right cost and available for the right period"*

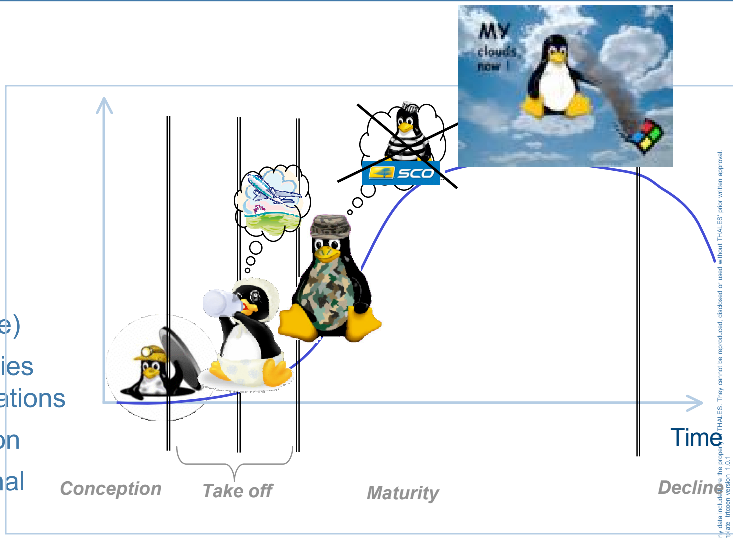
Approach

- Technical assessment
- Industrial assessment

OSS dedicated Evaluation process

Investigation

- Project (life)
- Communities and motivations
- Competition
- Professional support market



OSS dedicated Evaluation process

Emergence
- Initial objective
- Initiators
- Origin of the project
- Positioning
- 1st stable release

Taking-off
- Major events
- Successive stable releases

Maturity

Understanding
- Current model of development
- Developers community
- Positioning
- Adherence to standards
- Licensing scheme

Support
- Mailing lists
- FAQ
- Forums
- Commercial companies ? costs ?

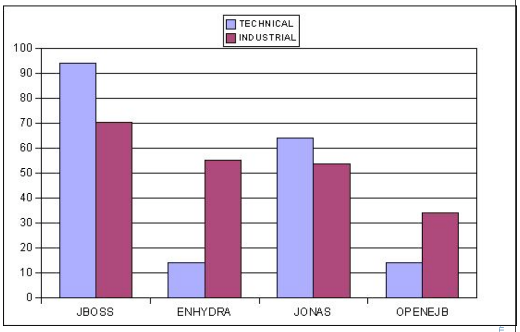
Acceptance
- Echo in the press
- Related web sites
- Industrials supporting the development
- Number of users
- Institutional & industrials users

Future
- Roadmaps

Evaluation - J2EE sample

Technical (aggregated) criterion & Weight

EJB 2.0 support	**50**
CMP 2.0 support	**30**
Database support	**20**
Total	*100*



Industrial (aggregated) criterion & Weight

Professional technical support	**20**
Users population (nb, role)	**15**
Project (re-)activity (Q/A Mailing lists)	**13**
Release & correction frequency	**12**
Company hosting	**10**
Developers community	**8**
Information (Web & Forums)	**7**
Documentation	**6**
Relationships with other OSS	**5**
Press/web footprint	**4**
Total	*100*

Process run & assessed on 2002 (MILOS project)
- 350 OSS (45 segments)
- Results published on **eCOTS portal**

Part 1 : Conclusion

Introducing OSS in the scope is a change,
then, organizational changes are needed

- A corporate policy to control correct usage of OSS
- A dedicated team (multi units)
  - to provide legal analysis, advice and audits
  - to capitalize and to organize technical exchanges (workshops, lessons learnt) ; to set up networks of experts (evaluation process)
  - to make known issues (awareness campaigns to all stakeholders)
  - to ensure the smooth running of local organization (enterprise level) in charge of procurement, validation/qualification, deployment, configuration management
  - to survey external expertise (support market)
- Updates of corporate referential
  - Components evaluation & selection guideline
  - Components usage guideline

How to benefit from OSS:

- through the collaborative development model
- through the process

Benefit from the OSS development model

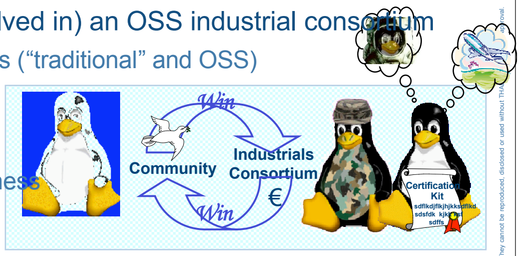
A model to improve quality, productivity & collective intelligence

- Thales Collaborative Development Platform
  - Experimentation started mid 2002
  - 25 projects - 60 active developers
- Experimentation assessment on-going, preliminary results show benefits related to:
  - Reuse (& convergence)
  - Synergy - Sharing & co-operation
  - Quality (peer reviews)
  - Personal motivations (recognition)
  - Technological communities
  - Unified project referential for all artefacts (source, doc, mail, news, bug tracking…)

A great disruption : from local (department/division/unit) to corporate interest

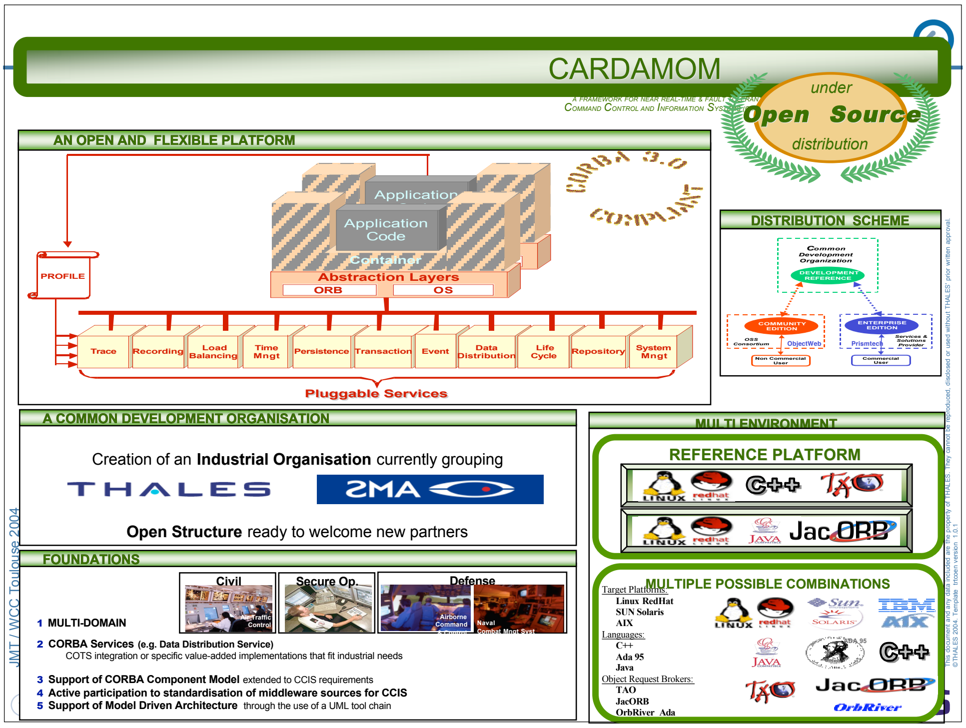
A mean towards open architectures and standardization
perpetuate R&D investment

- Launching of (or getting involved in) an OSS industrial consortium
  - take the best of the 2 worlds ("traditional" and OSS)



- Key success factors
  - Need covered & Attractiveness
  - Motivation
  - Licensing schema
  - Budget - Plan
  - Team building & Project management
  - Consortium building : Partnerships and strategic objectives
  - Community management

Make IP free (OSS) is not as easy as it could seem



Conclusion 1

OSS

- is there,
- is not a marginal phenomena
- comes with opportunity

- is disruptive :
  - changes in organizational structures : formal ones (units in charge to analyze risks, to recommend, to deliver, to maintain) and informal ones (networks of experts)
  - changes in organizational techniques (business referential) : purchase, business management, design, development integration, test/validation, deployment, maintenance

Use (**correctly**) when it makes sense
COTS & OSS have a place in systems

Conclusion 2

OSS allows :

- Improvement of performance
- Improvement of R&D investments
- Standardization (commodification) of software architecture
  (openness, interoperability, technology insertion)
- Sharing & cooperation on non-core business technology

OSS is **A** mean to meet our R&D strategy