Oberon day

Reliability in real-time: Why strong-typed programming languages matter

Roberto Brega, LogObject AG

Oberon day @ CERN, March 10th 2004

Contents

00 First-hand experience & personal background

Experience

- **1984 – 1990: The early years**
  - Commodore 64, basic, “GOTO” to be considered harmful [Dijkstra]
  - Pascal, compiler, epiphany: Indentation = poetry
  - Wirth N., Jensen K., Pascal: User Manual and Report
- **1990 – 1996: Dept. Informatik, ETH Zürich, Computer science**
  - Oberon language used for teaching basic computer science courses, system software, compiler construction, electronics (FPGA programming, LOLA), semester- and diploma-thesis
  - Internship at Ubilab: C++, Java
- **1996 – 2001: Institute of Robotics, ETH Zürich, Ph.D.**
  - Assistantship, research

Experience

- Robotics (mechatronics) = Computer science + Electronics + Mechanics
- Robotics = Cool, cult-status among students (physics and maths, too)
- Result: Huge student audience, lousy programmers ;)

- How can we harness student man-power without undermining research and industry projects?
- How can we let non-programmers quickly grasp the tools they need?
- How can we let non-CS engineers focus on the problem to be solved?
- How can I marry research topic with the assistantship duties?

```
=    "(" [FPSection {";" FPSection}] ")" [":" Qualident].
=    [VAR] ident {"," ident} ":" Type.
=    "(" [VAR] ident ":" ident ")".
=    Qualident
|    ARRAY [ConstExpr {"," ConstExpr}] OF Type
|    RECORD ["("Qualident")"] FieldList {";" FieldList} END
|    POINTER TO Type
|    PROCEDURE [FormalPars].
=    [IdentList ":" Type].
=    Statement {";" Statement}.
=   [  Designator ":=" Expr
|     Designator ["(" [ExprList] ")"]
|     IF Expr THEN StatementSeq {ELSIF Expr THEN StatementSeq} [ELSE Statement
|     CASE Expr OF Case {"|" Case} [ELSE StatementSeq] END
|     WHILE Expr DO StatementSeq END
|     REPEAT StatementSeq UNTIL Expr
|     FOR ident ":=" Expr TO Expr [BY ConstExpr] DO StatementSeq END
|     LOOP StatementSeq END
|     WITH Guard DO StatementSeq {"|" Guard DO StatementSeq} [ELSE Statement
|     EXIT
|     RETURN [Expr]
    ].
=    [CaseLabels {"," CaseLabels} ":" StatementSeq].
```

Oberon-2 syntax (EBNF, Excerpt)

Experience

Appendix B: Syntax of Oberon

```
Module        =  MODULE ident ";" [ImportList] DeclSeq  [BEGIN StatementSeq] END ident ".".
ImportList    =  IMPORT [ident ":="] ident {"," [ident ":="] ident} ";".
DeclSeq       =  { CONST {ConstDecl ";" } | TYPE {TypeDecl ";"} | VAR {VarDecl ";"}} {ProcDecl ";" | ForwardDecl ";"}.
ConstDecl     =  IdentDef "=" ConstExpr.
TypeDecl      =  IdentDef "=" Type.
VarDecl       =  IdentList ":" Type.
ProcDecl      =  PROCEDURE [Receiver] IdentDef [FormalPars] ";" DeclSeq [BEGIN StatementSeq] END ident.
ForwardDecl   =  PROCEDURE "↑" [Receiver] IdentDef [FormalPars].
FormalPars    =  "(" [FPSection {";" FPSection}] ")" [":" Qualident].
FPSection     =  [VAR] ident {"," ident} ":" Type.
Receiver      =  "(" [VAR] ident ":" ident ")".
Type          =  Qualident
              |  ARRAY [ConstExpr {"," ConstExpr}] OF Type
              |  RECORD ["("Qualident")"] FieldList {";" FieldList} END
              |  POINTER TO Type
              |  PROCEDURE [FormalPars].
FieldList     =  [IdentList ":" Type].
StatementSeq  =  Statement {";" Statement}.
Statement     =  [  Designator ":=" Expr
              |   Designator ["(" [ExprList] ")"]
              |   IF Expr THEN StatementSeq {ELSIF Expr THEN StatementSeq} [ELSE StatementSeq] END
              |   CASE Expr OF Case {"|" Case} [ELSE StatementSeq] END
              |   WHILE Expr DO StatementSeq END
              |   REPEAT StatementSeq UNTIL Expr
              |   FOR ident ":=" Expr TO Expr [BY ConstExpr] DO StatementSeq END
              |   LOOP StatementSeq END
              |   WITH Guard DO StatementSeq {"|" Guard DO StatementSeq} [ELSE StatementSeq] END
              |   EXIT
              |   RETURN [Expr]
                 ].
Case          =  [CaseLabels {"," CaseLabels} ":" StatementSeq].
CaseLabels    =  ConstExpr [".." ConstExpr].
Guard         =  Qualident ":" Qualident.
ConstExpr     =  Expr.
Expr          =  SimpleExpr [Relation SimpleExpr].
SimpleExpr    =  ["+" | "-"] Term {AddOp Term}.
Term          =  Factor {MulOp Factor}.
Factor        =  Designator ["(" [ExprList] ")"] | number | character | string | NIL | Set | "(" Expr ")" | " ~ " Factor.
Set           =  "{" [Element {"," Element}] "}".
Element       =  Expr [".." Expr].
Relation      =  "=" | "#" | "<" | "<=" | ">" | ">=" | IN | IS.
AddOp         =  "+" | "-" | OR.
MulOp         =  " * " | "/" | DIV | MOD | "&".
Designator    =  Qualident {"." ident | "[" ExprList "]" | " ↑ " | "(" Qualident ")"}.
ExprList      =  Expr {"," Expr}.
IdentList     =  IdentDef {"," IdentDef}.
Qualident     =  [ident "."] ident.
IdentDef      =  ident [" * " | "-"].
```

Oberon-2 syntax (EBNF, Full)

XO/2

01

XO/2, an RTOS based on the programming language Oberon

XO/2

REAL-TIME OS

XO/2 real-time operating system

XO/2 is an object-oriented, hard-real time system software and framework, designed for safety, extensibility and abstraction. It takes care of many common issues faced by programmers of mechatronic products.

Features

- Written in the Oberon-2, type-safe, object-oriented programming language
- Deadline-driven scheduler with admission testing
- High frequency RT-scheduler (10 KHz) with 0.5% overhead on PowerPC 604@300MHz
- Full MMU support with paging on light-weight threads
- Real-time, incremental garbage collector with zero memory requirement during traversal
- Safe dynamic linking-loading and unloading
- WCET estimator, by means of PM data

Oberon language

- The choice was not an afterthought, rather central to the safety concepts
- Easy to learn (students) and to write a compiler for (system programmers)
- Powerful enough for imperative, modular, object- or component-oriented programming
- One of the few languages that mandates:
  - Typing information verified at compile- and run-time
  - Module’s interfaces, dynamically checked against at compile- and linking-time
  - Automatic memory reclamation (garbage collection)

XO/2

Deadline-driven scheduler

- No user-definable interrupts
- Each real-time task is installed by *duration*, *deadline*, *period*
- Scheduler tests timing constraints at admission time
- Scheduler monitors timing constraints at run-time

Real-time compatible dynamic memory reclamation (garbage collection)

- Heavily modified mark-and-sweep
- Deterministic penalty on memory access (compiler emitted code)
- Full heap-traversal without tasks-interruption
- Full heap-traversal without memory-requirements

Worst-case execution time (WCET) approximation

- High-performance processor architectures are non-deterministic (timing requirements)
  - Deep pipelines, super-scalarity, branch-prediction, deep caches, etc.
  - Object-oriented programming introduces late-binding
  - Pre-emption introduces non-foreseeable delays
- XO/2 WCET estimate
  - Compiler annotations
  - Run with performance monitoring hardware (event counters)
  - Compile with RT-statistics yields estimate
- Estimate: -5% < worst-case < +10%

XO/2

End-user restrictions

- No untyped memory accesses
- No manual memory management
- No software or hardware interrupts
- Pedantic compiler
- Bottom line: no cheating!

X O/2

End-user benefits

These compile-time and run-time mechanisms, pervasive yet efficient, allow the system to maintain a deus ex-machina knowledge about the running applications. The application programmer, relieved from many computer-science issues, can better focus his attention to the actual problem to be solved.

02 Deployment examples, applications in the embedded world

LOTraffic

LOTraffic is a sensor system aimed at pervasive traffic monitoring, control and enforcement. It has been recently received government approval (METAS) for speed ticketing. (Photo courtesy CES AG, Dübendorf)

LOTraffic

LOTraffic is a sensor system aimed at pervasive traffic monitoring, control and enforcement. It has been recently received government approval (METAS) for speed ticketing. (Photo courtesy CES AG, Dübendorf)

In-vivo measurement of human tissues

Laparoscopic instrument for the sampling elasto-mechanical properties of live organs. In-vivo measurements on the human uterus have been performed during regular hysterectomy interventions. (Photo courtesy Institute of Robotics, ETHZ)

Meyco RoboJet

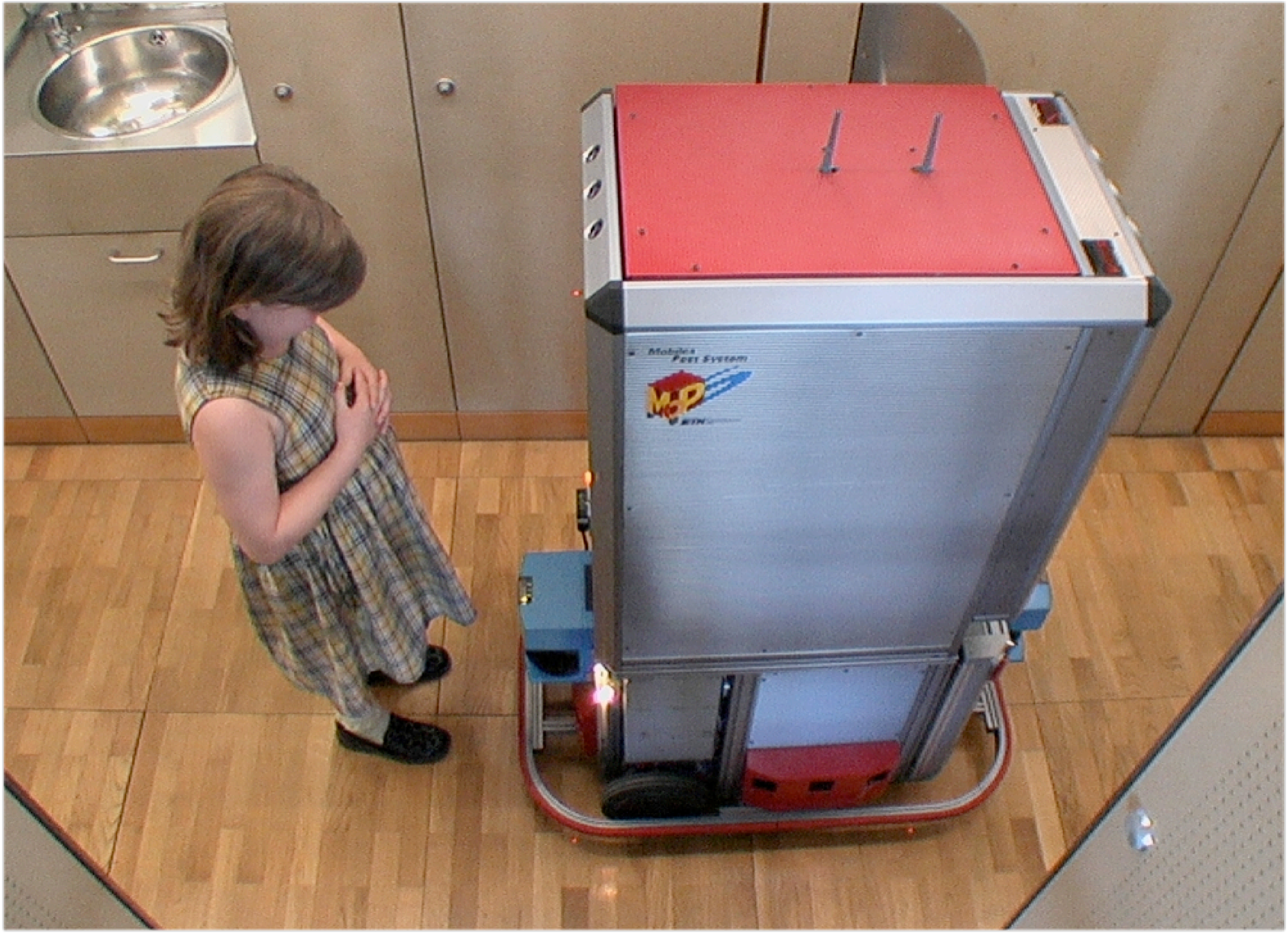
The RoboJet is an hydraulically actuated manipulator used in tunnelling construction work. Its task consists of spraying liquid concrete on the walls of new tunnels using a jet as its tool. The calculation of the redundant inverse kinematics and the closed-loop control of the 8 hydraulic actuators is performed by the control system in real-time.

(Photo courtesy Institute of Robotics, ETHZ & Meyco AG)

Examples

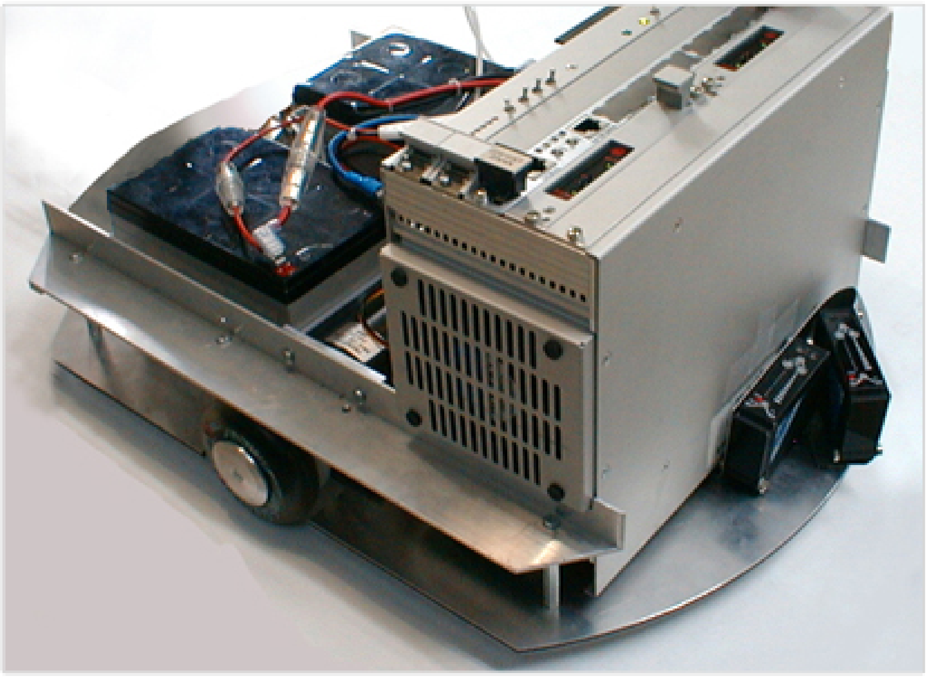
Anæsthesia Control System

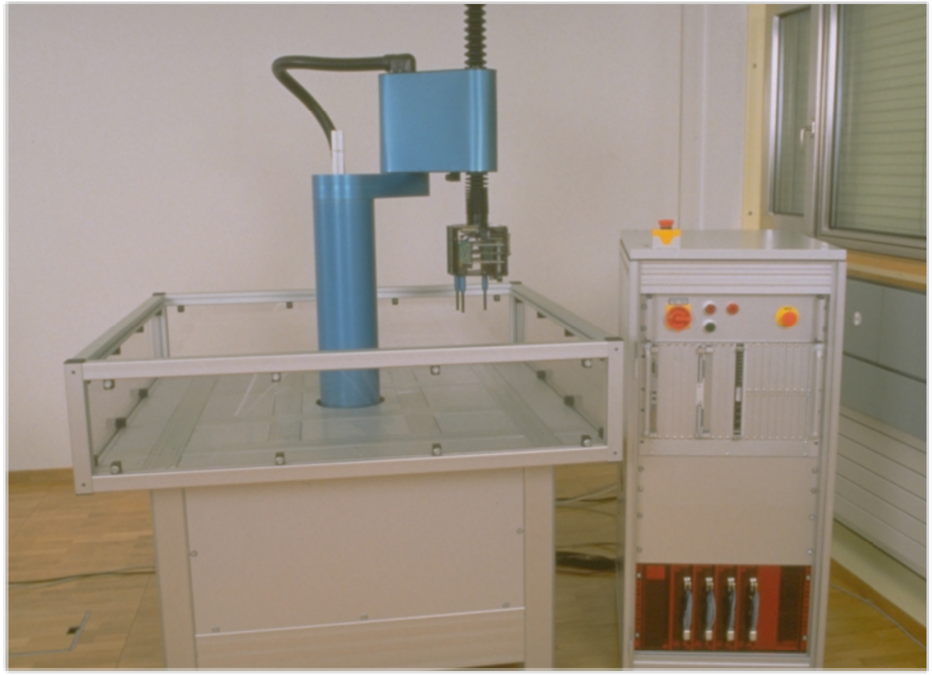
The Anæsthesia Control System is a closed-loop automatic-control for hypnosis.
(Photo courtesy Automatic Control Laboratory, ETHZ)

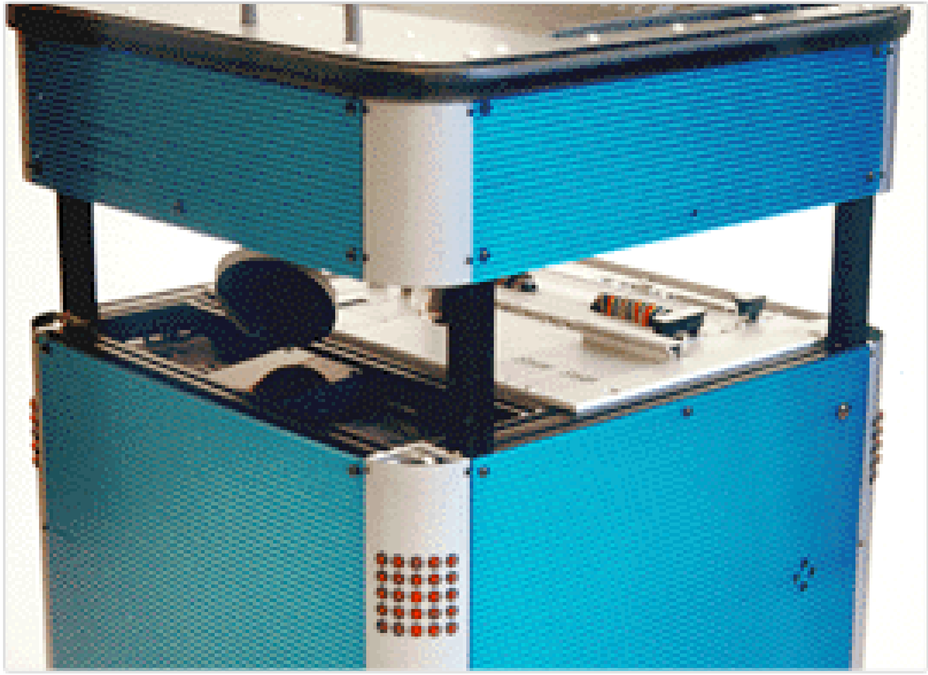


MoPS: Mobile mail distribution system

The Mobile Mail Distribution System MoPS represented a milestone in the fields of autonomous navigation and localisation. (Photo courtesy Institute of Robotics, ETHZ)



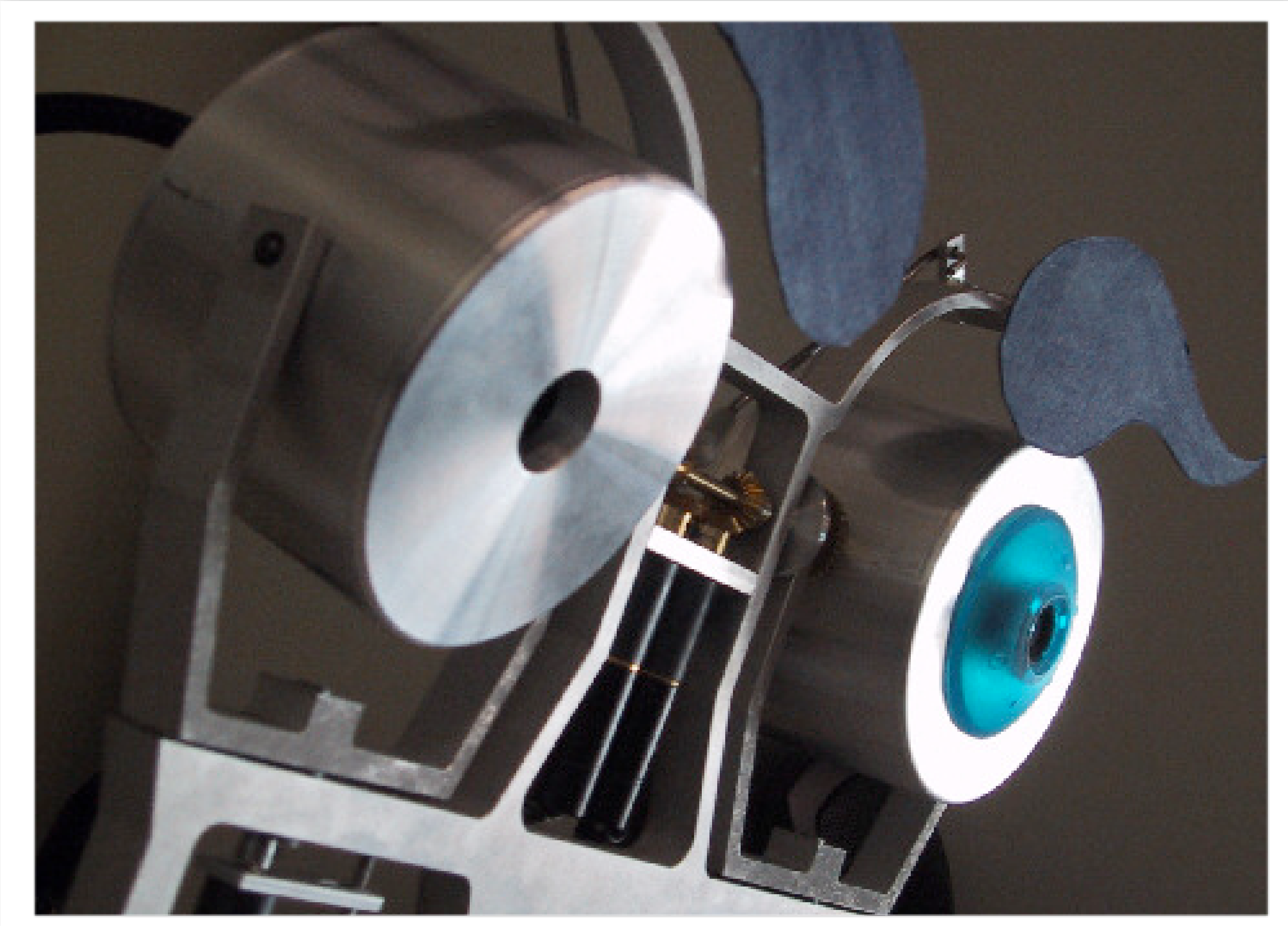




Further examples

SmartROB-II: Mobile robotics development platform; Hexaglide: High-speed, parallel milling machine; Inter/Milan: Low-cost, SCARA-type manipulator; Pygmalion: Mobile robot for autonomous map-building.

Examples



Robotics@expo.02

Photo courtesy ASL, EPFL & BlueBotics AG

03 Case-study: Robotics@ expo.02

Robotics@expo.02

Robotics@expo.02 was a very successful project presented at Expo.02—the Swiss National Exhibition in Neuchâtel. (Photo courtesy ASL, EPFL & BlueBotics AG)

Goals

- Project had to convey the feeling of increasing closeness between human and machine.
- Visitors had to be able to interact with up to eleven autonomous, freely navigating tour guide robots.
- From design to deployment in 18 months.
- Artistic experience coupled with 14 hours/day uptime.
- (Most of the) project members had no strong CS-background.
- Implemented, tested code base close to zero.

Two tasks, two groups /Navigation

- Classical mobile robotics, real-time requirements
  - Mobile platform control (drivers, ...)
  - Localisation, navigation (math)
  - Obstacle avoidance (math)
  - Mission control (big FSM)
- Deployed on an industrial, PowerPC-based platform
- Software written in Oberon, on top of XO/2

Two tasks, two groups /Interaction

- Robotics interaction
  - Peripheral control (drivers, ...)
  - Face recognition (math)
  - Voice output (libraries)
  - Interaction control (big FSM)
- Deployed on an industrial, Intel-based platform
- Software written in C/C++, on top of Microsoft Windows 2000

Variable	Navigation	Interaction
Team [persons]	4	6
Total work [man-years]	4 + 1 (re-use)	5
Micro-eng. [man-years]	1.5	3
Electronics-eng. [man-years]	1.5 + 0.5 (re-use)	1
CS-eng. [man-years]	1 + 0.5 (re-use)	1
Compile code [KB]	1376	1703

Robotics@expo.02: Groups breakdown

Case-study

Run-time	13’313 h
Movement time	9’415 h
Travelled distance	3’316 km
Failures (total/critical/non-critical)	4’378 / 4’086 / 292
Critical SW failures (Interaction, Navigation)	3’216 / 694

Robotics@expo.02: Statistics

Interaction failures

- Memory errors: Spurious pointer references; pointer arithmetic overflows; arithmetic operations on non-initialised, null, or invalid pointer; read/write operations through non-initialised or null pointers; procedure calls through non-initialised, null or invalid pointers; wrong type casting; array references out of declared bounds and non-initialised array index. Furthermore, memory leaks and dangling pointers have harmed the reliable run-time of the application.

- Others

Navigation failures

- Lost situations: In some cases, the robots were unable to determine their location and thus notified a lost situation.
- Others: Deadline violations, out-of-memory conditions, etc.

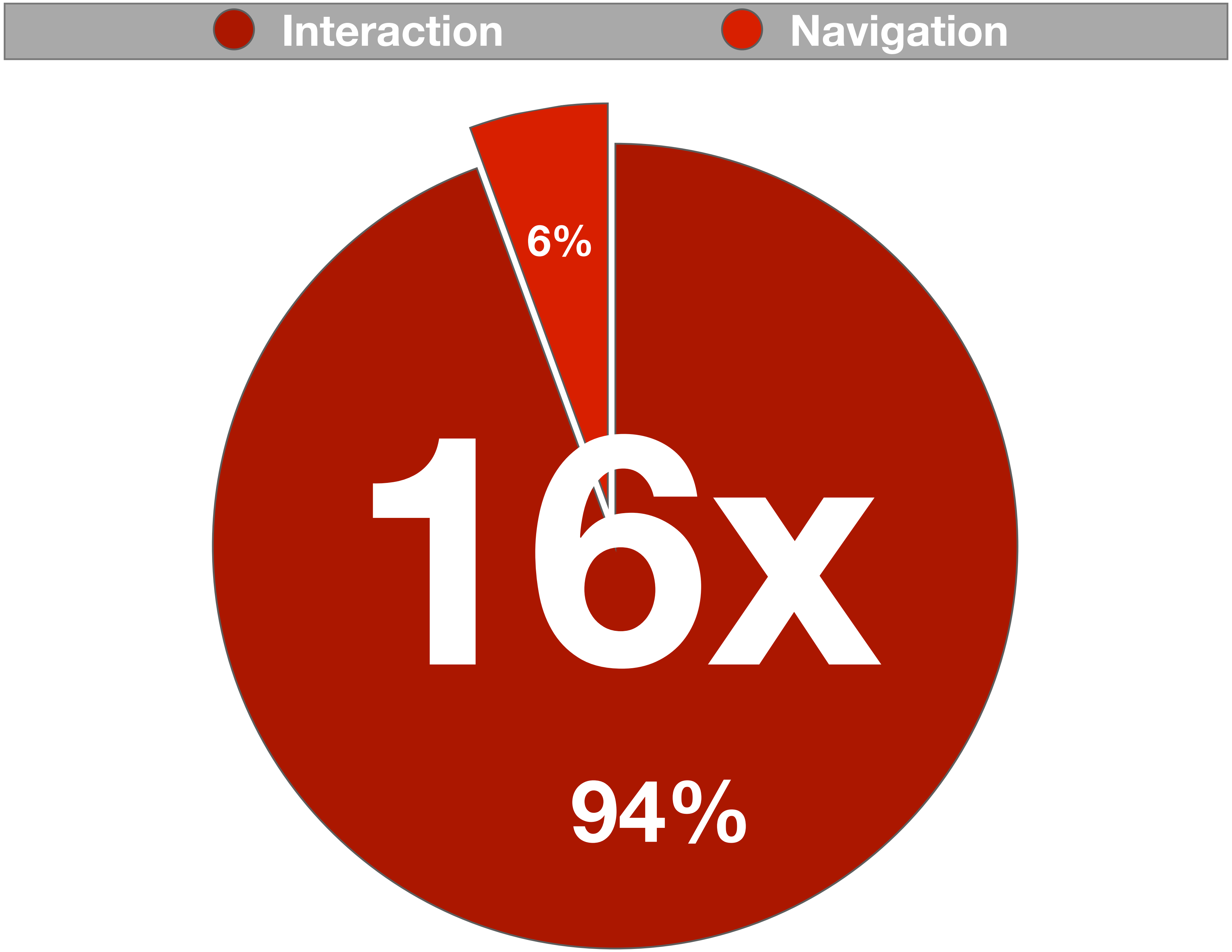
Robotics@expo.02: Navigation Failures

Case-study

Case-study

Run-time	13’313 h
Movement time	9’415 h
Travelled distance	3’316 km
Failures (total/critical/non-critical)	4’378 / 4’086 / 292
Critical SW failures (Interaction, Navigation)	3’216 / 190

Robotics@expo.02: Statistics



Robotics@expo.02: Interaction vs. Navigation failures

Case-study

04 Final remarks, wishes, hopes

Commenting the results

- The expo.02 example is not definitive, non-scientific, informal.
- That being said, all things being more or less equal, we have 16-times more failures in the interaction part.
- Maybe it is the Intel x86? Or Windows? Or bad programmers on one side only?
- We argue the missing type-safety of C/C++ is the culprit.

May you have a strong(-typed) foundation

- When facing a choice, choose strong-typing over loose-typing: The time you spend in letting the compiler accept your code will pay off at run-time!
- When evaluating the penalties imposed by automatic memory reclamation, think about debugging manual memory disposal: No OOP without GC!
- Guidelines are great (in theory) but are undermined by schedules, laziness, the weakest-link in the programming chain. It will not work!
- Safety is not an afterthought!
- Efficiency & semantics go hand-in-hand!

Java is great!

- Imitation is the sincerest form of flattery!
- Some design decisions are weird:
  - Bytecode is bad: pCode, been there done that! (N. Wirth)
  - Assembly is not the best intermediate representation for optimizing compilers (M. Franz)
  - OOP is not the silver bullet! (F.P. Brooks)
- Some design decisions are good:
  - Powerful, expressive interfaces
  - Exceptions

Dear Niklaus (Prof. Wirth)

- For **Oberon.next**, I would like:
  - Programming by contract (Eiffel)
  - TRY-CATCH clauses (Java)
  - More expressive interfaces: ABSTRACT, FINAL, etc.
  - Object finalization (orthogonal, unlike constructors)

0A Appendix: Movies, photos & references

Deadline-driven scheduler

- For synchronous periodic tasks (Lyu, Lailand)

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1$$

Deadline-driven scheduler

- For hybrid (aperiodic, asynchronous) tasks

Analyse (τ):

IF $U > 1$ THEN RETURN "Not Admitted" END;

$$t_A = \max\left\{D_{max}, \frac{\sum_{i=1}^{n}(1 - D_i/T_i)C_i}{1-U}\right\};$$

$$L = \text{Synchronous busy period length};$$

$$t_{max} = \min\{t_A, L\};$$

$$S = \bigcup_{i=1}^{n}\{mT_i + D_i : m = 0, 1, \ldots\} = \{e_1, e_2, \ldots\};$$

$k = 1$;

WHILE $e_k < t_{max}$ DO

IF $h(e_k) < e_k$ THEN RETURN "Not Admitted" END;

$k = k + 1$

END;

RETURN "Admitted"

References

F.P. Brooks, "No Silver Bullet: Essence and Accidents of Software Engineering", Computer, Vol. 20, No. 4 (April 1987).

N. Wirth, “The Programming Language Oberon” (Revised Edition 1.10.90), Report 143, Nov. 1990.

C. Szyperski, “Insight Ethos: On Object Orientation in Operating Systems”, (PhD thesis; Swiss Federal Institute of Technology (ETH Zurich), Diss. No. 9884). vdf Hochschulverlag AG an der ETH Zürich, Zurich, Switzerland, 1992.

J. v. Ronne, A. Hartmann, W. Amme, and M. Franz; “Efficient Online Optimization by Utilizing Offline Analysis and the SafeTSA Representation”; in J. Powers and J. T. Waldron (Eds.), Recent Advances in Java Technology: Theory, Application, Implementation; Computer Science Press, Trinity College Dublin, Dublin, Ireland, ISBN 0-9544145-0-0, pp. 233-241; November 2002.

R. Brega, “A Combination of System Software Techniques Aimed at Raising the Run-Time Safety of Complex Mechatronic Applications”, Dissertation ETH Nr. 14513, Zürich, 2002.