The Quest for Autonomy: Programming Dependably Adaptive R/T Applications in CORTEX

Paulo Veríssimo
Univ. of Lisboa Faculty of Sciences
Lisboa – Portugal

pjv@di.fc.ul.pt
http://www.navigators.di.fc.ul.pt

49th Meeting of IFIP Working Group 10.4
Tucson, USA, February, 2006

Problem Motivation

- Design and deployment of distributed applications is faced with the confluence of antagonistic aims: *uncertainty vs. predictability*
- Current and future large, massive-scale pervasive and/or ubiquitous computing systems will amplify this
- Key lies with a changing notion of service guarantees, not with their absence

Dealing with uncertainty

- We defined a generic approach to reconcile uncertainty with the need for predictability:

→ Dependable adaptation

- Make the application behave [safely, timely, securely, etc] in the measure of what can be expected from the environment
- Provide some guarantees in the way you do that

Dependability framework for adaptability

Grand challenge put by this scenario?

withstanding uncertainty
whilst achieving predictability

- Reconciling them means:
  - Securing strong attributes in weak settings

  (where usually very little is assumed and very little is expected from)

- So far we had two philosophical pillars:
  - Binary notion of correct and incorrect
  - Weakening assumptions down to the point of getting to impossibility results

Guidelines

- Assume that *uncertainty is not ubiquitous* (and is not everlasting)--- the system has parts more predictable than others (and tends assume stable periods)
- Be *proactive in achieving predictability-*-- make it happen at the right time, right place
- *Tolerate uncertainty* further to tolerating faults--- not all failures can be prevented, and/or some only on a probabilistic basis

Back to the roots

- Initial idea (1999, later on IEEE TOCS 2002)
  - a hybrid system and architectural model
  - a programming model
  - some formal properties in the time domain
  - later extended to any fault space (e.g. security attacks)

- *Tolerate uncertainty* further to tolerating faults
  - not all failures can be prevented, and/or some only on a probabilistic basis

Dependability framework for adaptability

- Re-state ‘correctness’ definition:
  - Consider normal and critical properties
  - Normal properties can be violated within assumed bounds
  - Critical properties cannot

  (in fact a modern perspective on the weak-fail-silence notion in DELTA-4)

- In more formal terms:
  - *Given a system described by a set of critical properties P_C and a set of normal properties P_N, the system is correct iff any critical property is met with a coverage of one, and any normal property is met with a lower bounded coverage less than or equal to one*

Overarching predicates

- Generic predicates dictate system correctness, regardless of functional semantics
  - **Coverage Stability –** assumed coverage remains stable within bounds
  - **No-Contamination -** violation of normal properties never entails violation of critical properties

Predicates

No-Contamination: *Given a history* $\mathcal{H}(\mathcal{T}_{\mathcal{P}})$ *derived from property* $\mathcal{P} \in \mathcal{P}_A$, $\mathcal{H}$ *has no-contamination iff for any timing failure in any execution* $X \in \mathcal{H}$, *no safety property in* $\mathcal{P}_A$ *is violated.*

Coverage Stability: *Given a history* $\mathcal{H}(\mathcal{T}_{\mathcal{P}})$ *derived from property* $\mathcal{P} \in \mathcal{P}_A$, *with assumed coverage* $P_{\mathcal{P}}$, $\mathcal{H}$ *has coverage stability iff the set of executions contained in* $\mathcal{H}$ *is timely with a probability* $p_{\mathcal{H}}$, *such that* $|p_{\mathcal{H}} - P_{\mathcal{P}}| \leq p_{dev}$, *for* p_{dev} *known and bounded.*

Some fairly complete behaviour classes

- Define behaviour classes with regard to a property P:

- Adaptive
  - Recurrent violation of property P is accepted
    (with a known degree and/or probability)
- Safe
  - Occasional violation of property P is accepted
    (the system can react by F/T)
- Fail-safe
  - Any violation of property P is not acceptable
    (the system must stop)

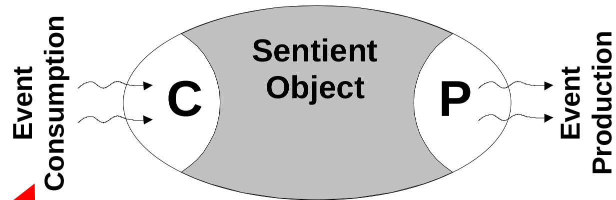
The Quest for Autonomy and Adaptability: CORTEX, 2001-2004

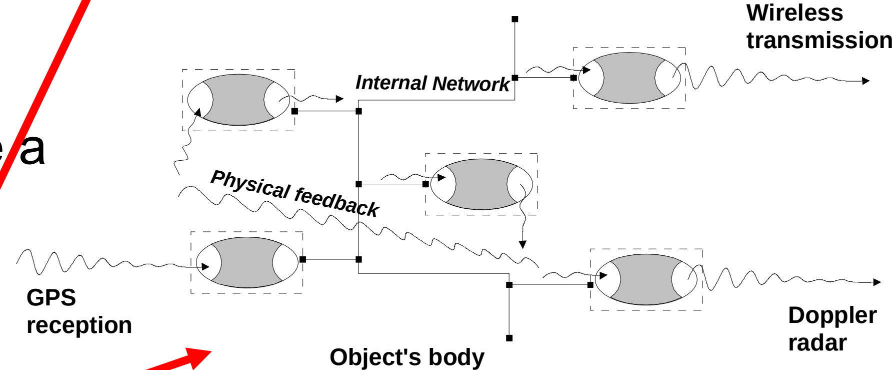
Context

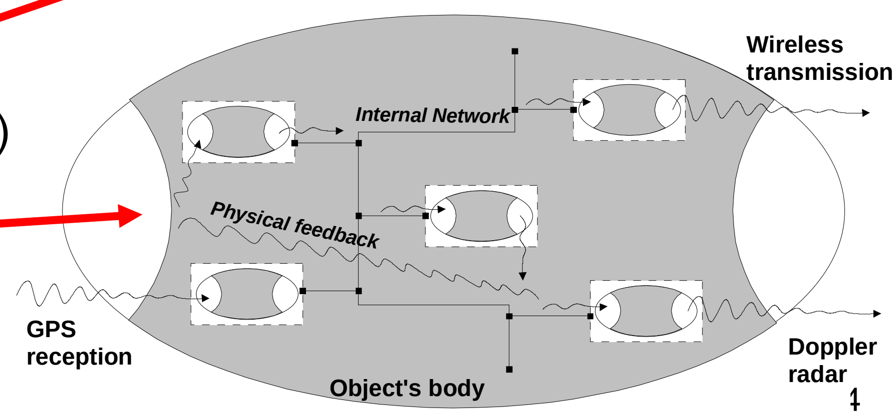
- Work developed in CORTEX, in which the concept of sentient objects was introduced:
  - Autonomous entities with sentience (e.g. robots)
  - Geographical dispersion, dynamic architecture
  - Real-time & safety & availability requirements
- Several issues addressed in CORTEX
  - Programming model for sentient applications
  - Enabling hybrid architecture (wormholes)
  - Interaction model featuring computer/environment fusion
  - WAN-of-CAN network architecture (systems-of-systems)

Sentient objects

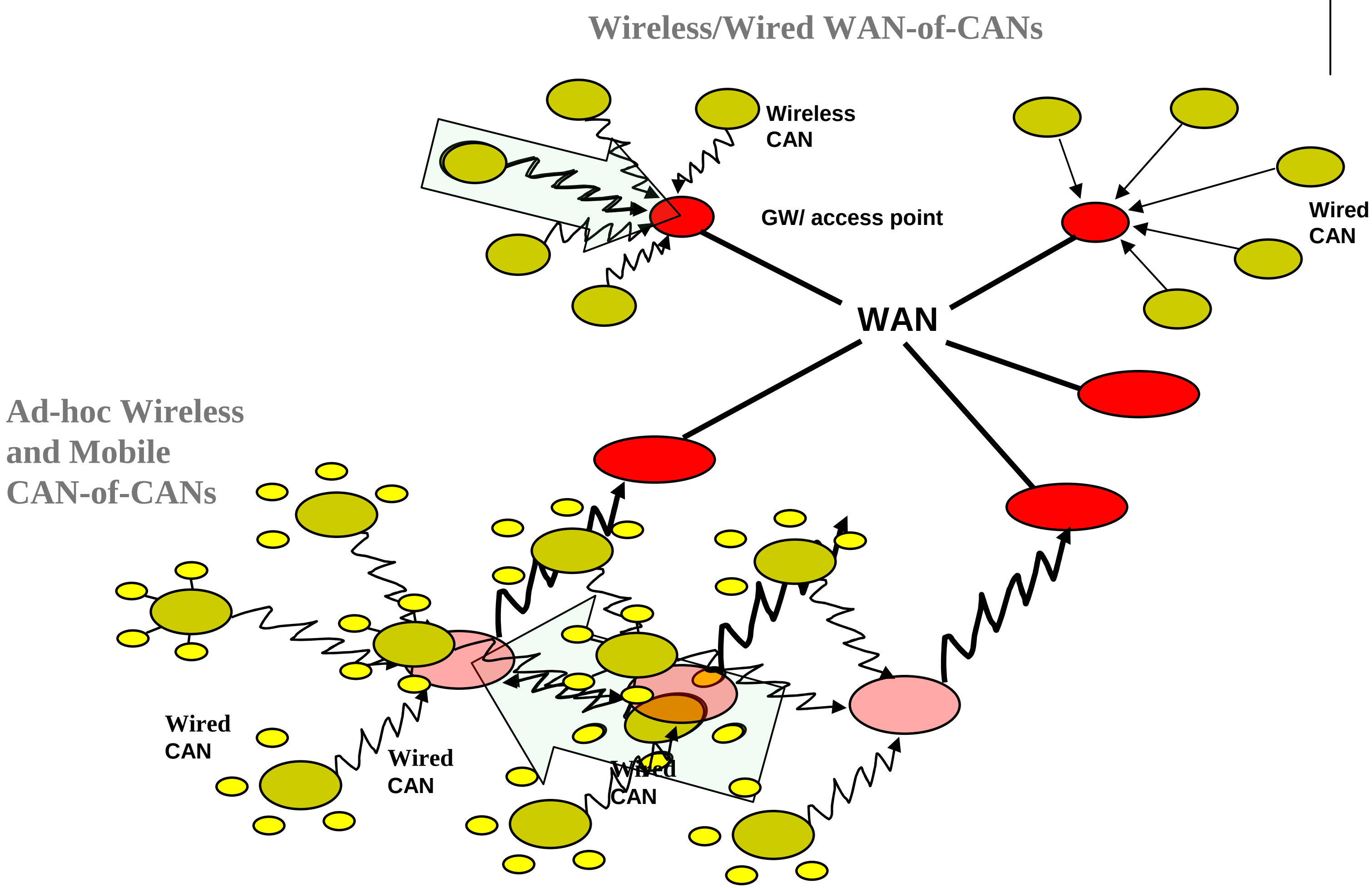
- Component-aware sentient object composition
  - clustering
  - hierarchical composition
  - normally constrained by the actual hardware component's structure
- To provide an example, imagine a robot and its manipulator controllers:
  - see each controller + control software as a sentient object
  - imagine structure and interconnections in the robot
  - see robot itself as a (composite) sentient object: the controller objects plus all the robot hardware (body)





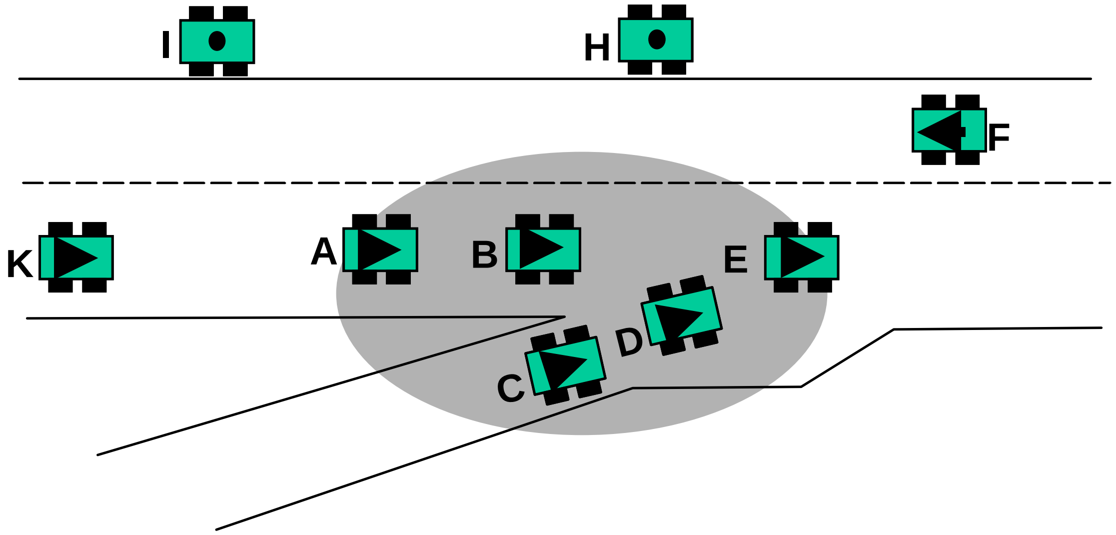


Networking: autonomy and mobility



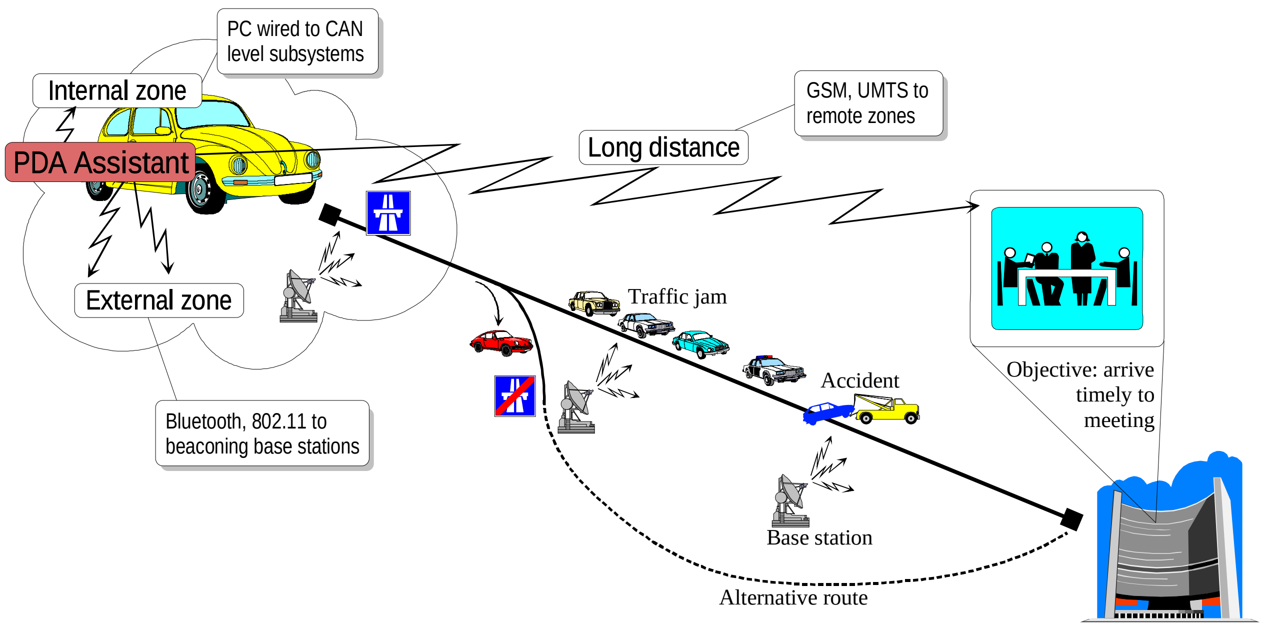
Application scenarios

- Cooperating Cars



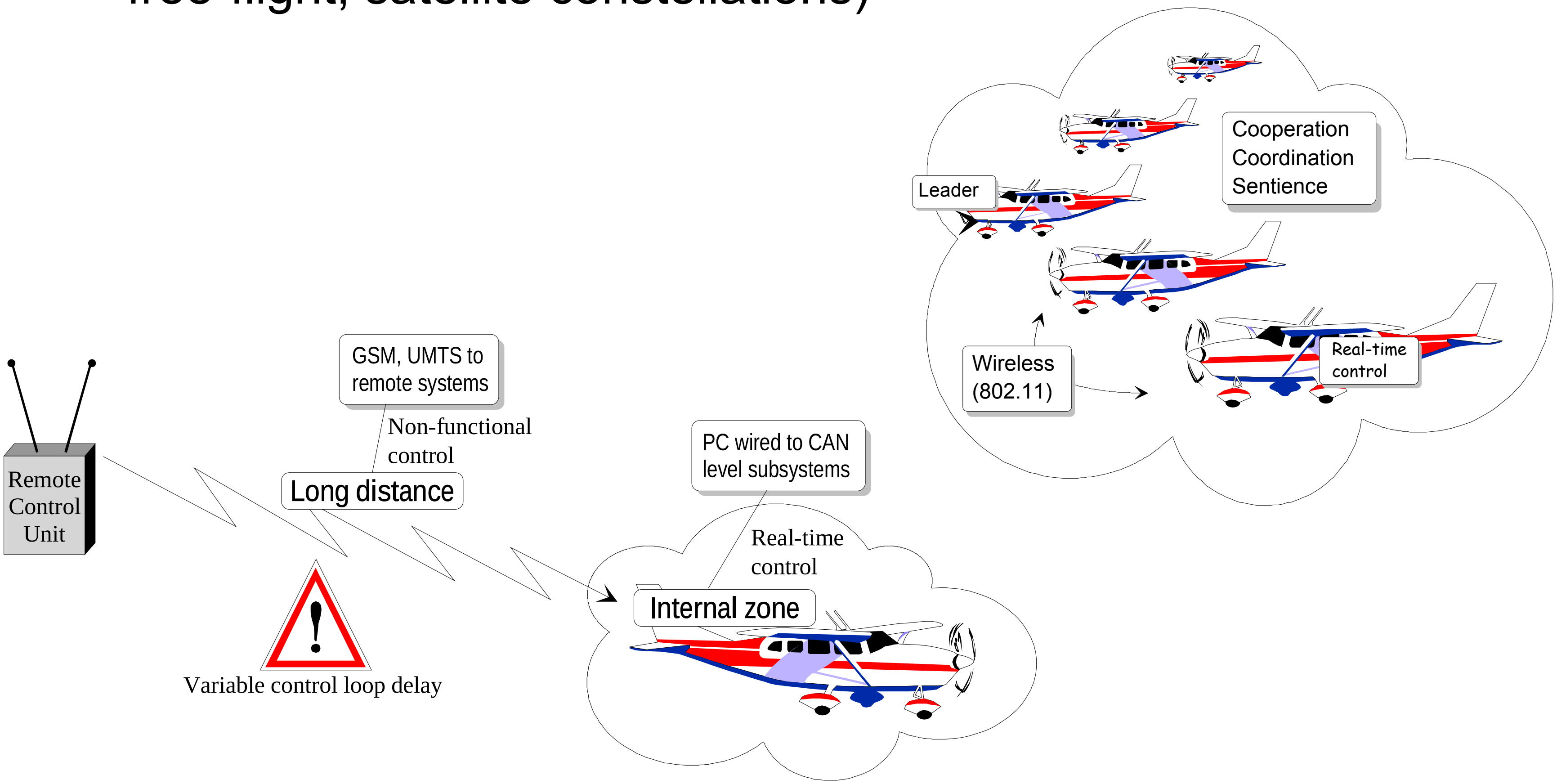
Application scenarios

- Assisted Terrestrial Transportation System
- Other wireless mobile gadget based ubiq. comp. appls



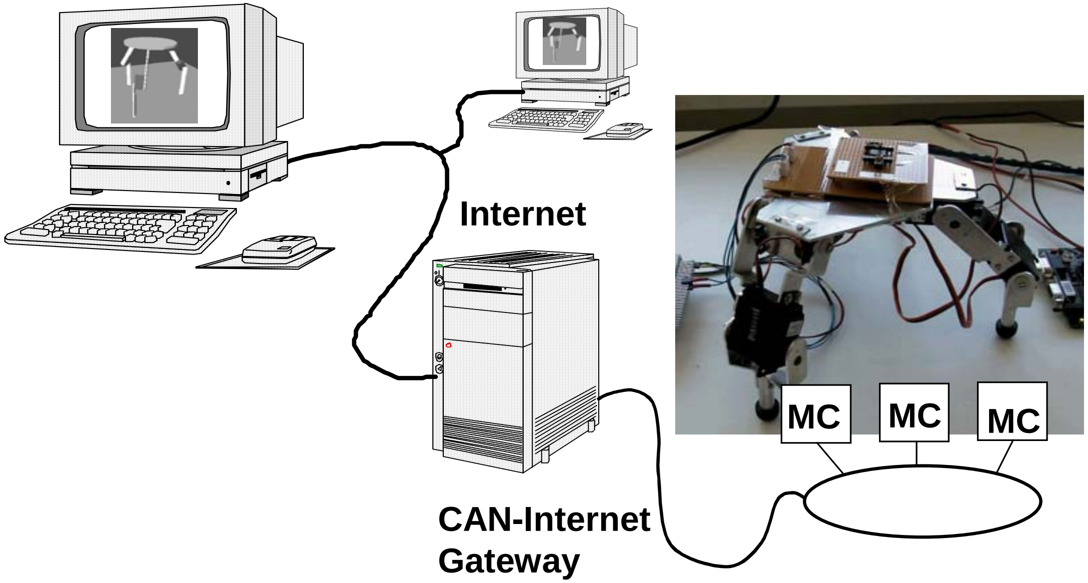
Application scenarios

- Autonomous or Remote control of real-time operations (e.g. free-flight, satellite constellations)



Application scenarios

- Remote control of a grabber robot
- Autonomous teams of robots or enhanced humans
- Other wireless mobile gadget based control or ubiq. comp. appls



A new Programming Model for Dependable Adaptive Real-Time Applications

Dependable adaptation at work
(time domain example)

- **Property classes** :
  - **Critical properties** - safety properties
  - **Normal properties** - timeliness properties
- **System predicates (in time domain)** :
  - **Coverage Stability -** coverage of timing assumptions remains stable within known error
  - **No-Contamination -** safety properties not violated on account of timeliness property violations

Dependable adaptation at work
(time domain example)

- Timing failures more complex than they look [IEEE TOCS 02]
  - **Unexpected delay** - "normal" effect
  - **Contamination** - error propagat. effect on safety props
  - **Decreased coverage** - continued occurrence effect

- Can we achieve correct operation despite these?
  - Contamination should be avoided (no-contamination)
  - Coverage should remain stable (coverage stability)

Dependable and Adaptive R/T Computing

- Introduced classes of system behaviour that deal with these problems in the time domain:
  - Fail-safe: correct behaviour or stops in fail-safe state [DSN2000]
  - Time-elastic: elastic time bounds with coverage stability [SRDS2001]
  - Time-safe: sporadic timing failures with no-contamination [DSN2002]
- Applied known fault tolerance techniques:
  - detection and/or recovery; masking
- NB: It is necessary to detect and react to timing failures:
  - **TIMING FAILURE DETECTOR (TFD) considered fundamental**

The TCB Model and Architecture [IEEE TOCS 02]

- **Application classes** :
  - **Time-Elastic -** Recurrent violation of a timeliness property is accepted, with a bounded probability
  - **Time-Safe -** Occasional violation of a timeliness property is accepted, its up to the system to react
  - **Fail-safe -** Any violation of a property is not acceptable, the system must stop

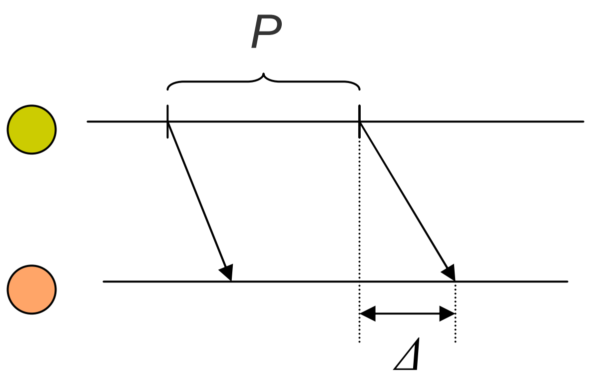
Example application frameworks

- Fail-safe operation [DSN2000] :
  - by switching to a fail-safe state after the first failure
  - requires the TFD service and appl´s to be of the fail-safe class
- Reconfiguration and adaptation [SRDS2001] :
  - by enforcing coverage stability
  - requires appl´s to be of the time-elastic and time-safe class
- Timing error masking [DSN2002] :
  - by using replication to mask transient timing errors
  - requires the TFD service and appl´s to be time-safe class

Classical approaches to R/T progr.

Consider a car driving control example: avoiding collision between two cars

- Traditional hard real-time approach is deadline-driven:



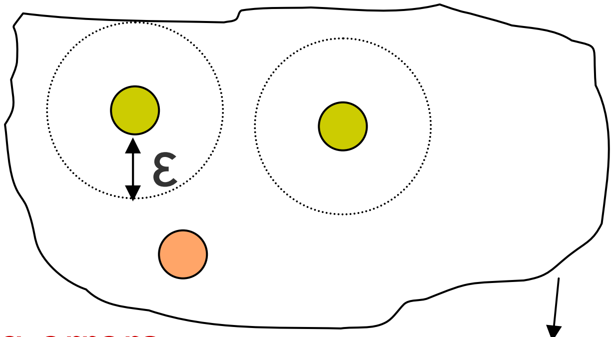
- Given target speeds, devise R/T schedule so that corrections made suffic. often.
- Static schedule loaded onto R/T executives
- Periodically, with a deadline of *P* units, cars exchange information and trajectory is corrected
- Missed deadline is a failure in HR/T system

- Consequence:
- The deadline became the goal
- The safety distance became accessory

Our approach to R/T progr.

Consider a car driving control example: avoiding collision between two cars

- Our approach:
- SAFETY DISTANCE Property: A car cannot “enter” the dashed circles of other cars, i.e must remain at a distance ε
- Each car must know other cars’ positions with a bounded error
- Distance ε proportional to the error
- Error depends on physics (fixed) and on period and delay of comm’s (variable)
- Allowed speed proportional to ε



Orange car’s view of the environment

- Consequence:
- The safety distance is the goal
- The speed and deadlines are accessory
- They become timed actions, which can have timing errors,
- Errors can be handled by timing fault tolerance

Programming principles

- General and systematic approach:
  - Timing failure detection service
    - Provide a bound for some action
    - Execute a handler upon failure detection
  - QoS coverage service
    - Assure needed coverage for each timing variable
    - Automatic adaptation of the variable
    - For applications with time-safety and time-elasticity

Making it dependable

- To adapt the QoS it is necessary to:
  - monitor the actual QoS being provided
  - decide if adaptation is necessary

- To dependably adapt the QoS we must:
  - observe the environment in a dependable way
  - apply a rigorous strategy about when and how to adapt

Making it dependable

- To avoid contamination it is necessary to:
  - prevent timing failures from propagating effects to safety properties

- To confine timing errors we must:
  - detect timing failures *timely*
  - apply a rigorous strategy about how to react

Dependable adaptation

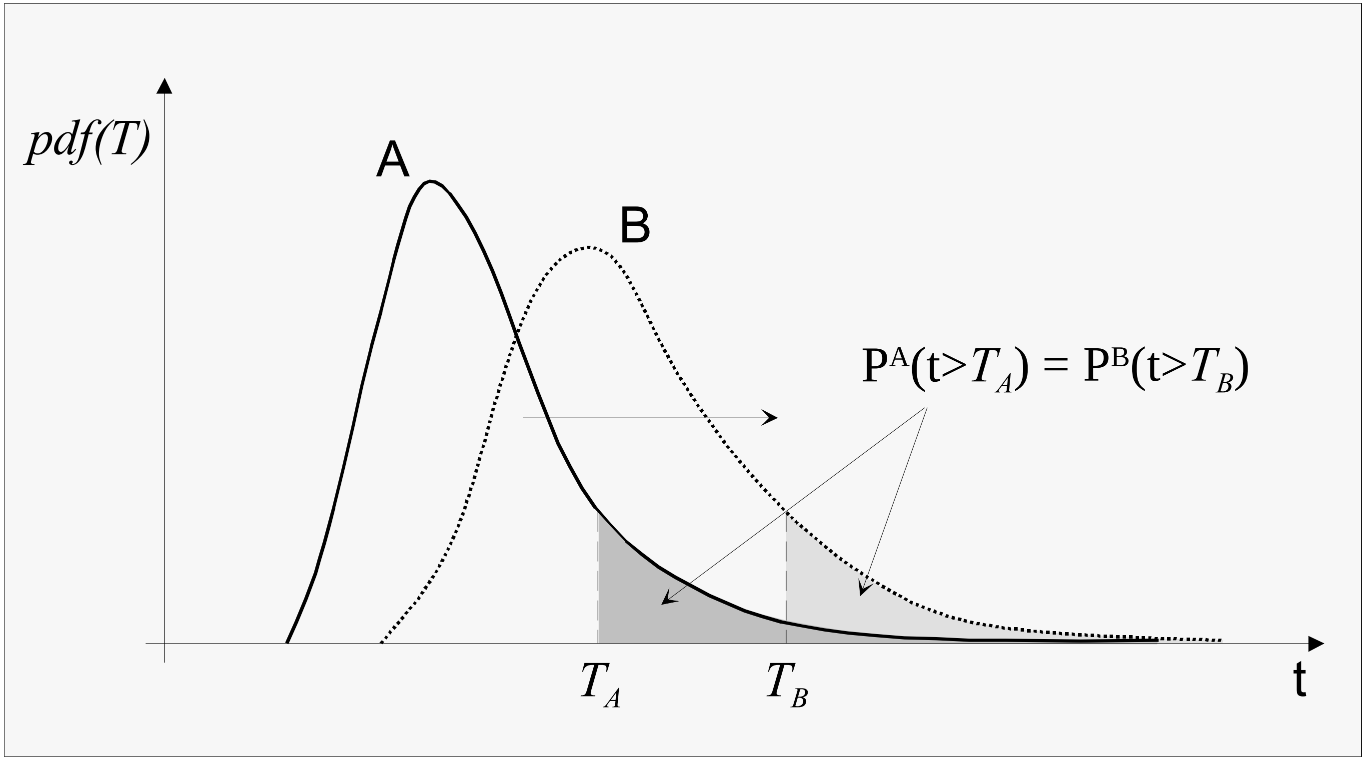
- It is necessary to trust the service that provides the measurements (durations)
  - in the value domain (correct measurements)…
  - …and in the time domain (timely measurements)

Timing failure detector

- **Timed Strong Completeness**
  - *There exists $TTFD_{max}$ such that given a timing failure at p in any timed action, it is detected within $TTFD_{max}$ from t_e*
- **Timed Strong Accuracy**
  - *There exists $TTFD_{min}$ such that any timely timed action that does not terminate within $-TTFD_{min}$ from t_e is considered timely*

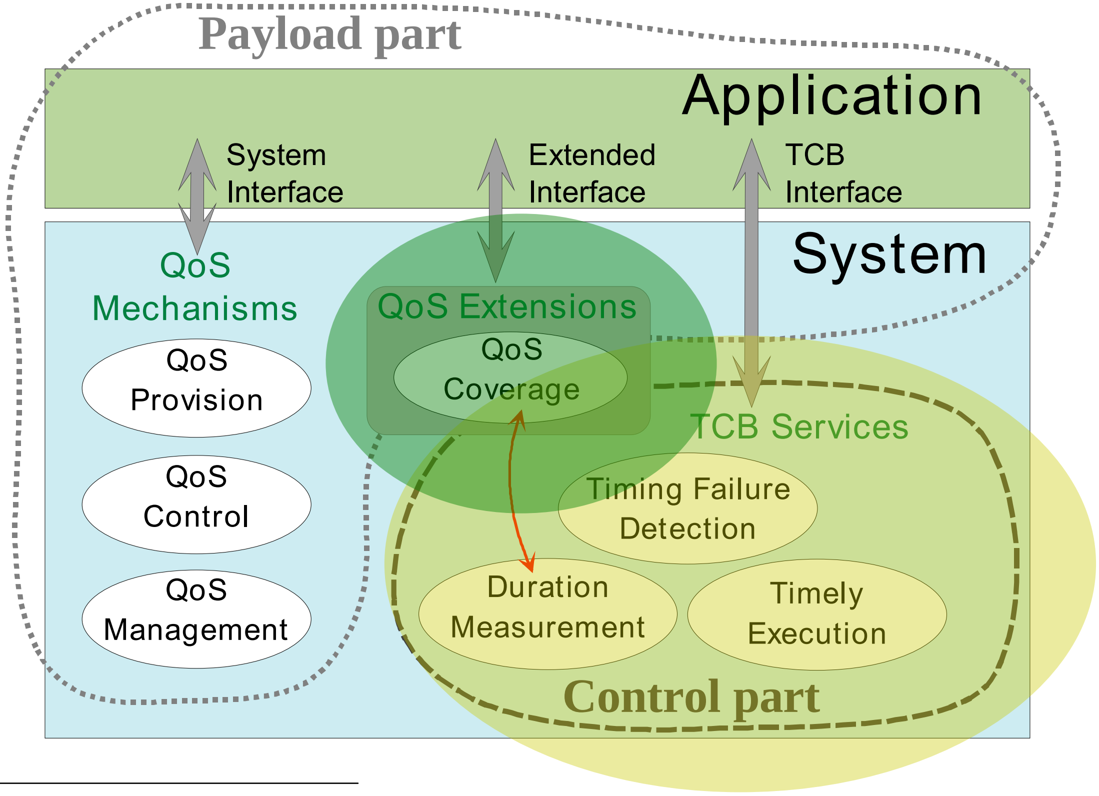
Dependable adaptation

- Then, decide when and how to adapt



QoS coverage service

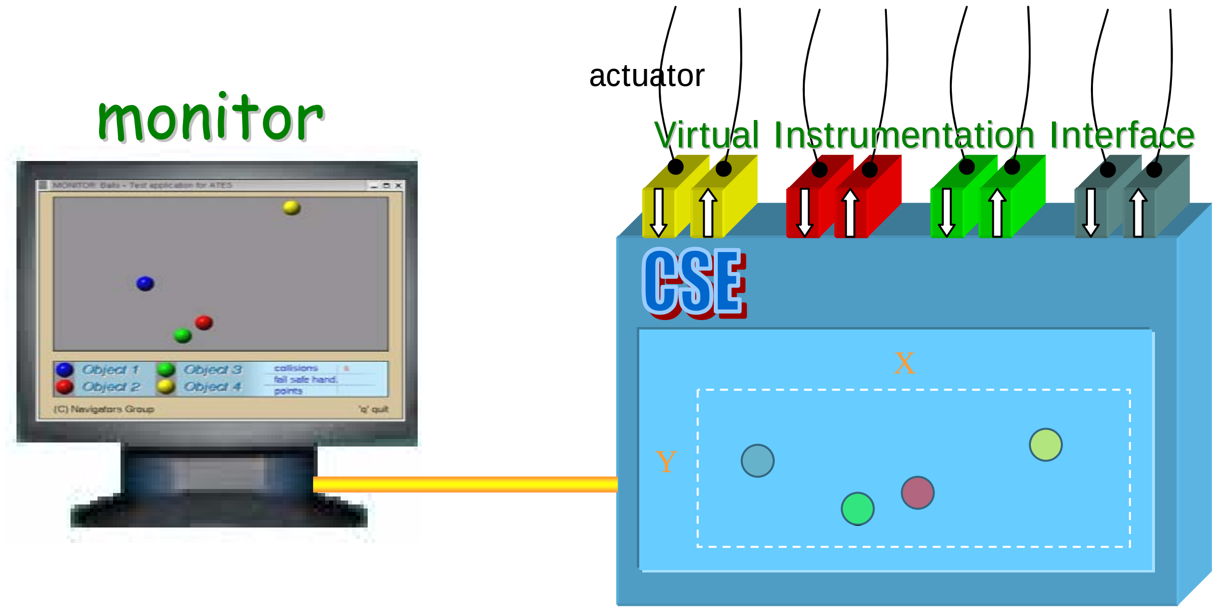
Example of a system with a TCB wormhole



Applying the programming model

Sentient objects emulator

- Emulates physical environments in real-time



Emulator

- Emulated environment: four entities shaped as colored balls move in a space with a certain speed and direction
- A Virtual Instrumentation Interface allows to:
  - acquire ball positions, directions and speeds;
  - change ball movement (speed and direction)
- Uses the TCB for the underlying services:
  - QoS Adaptation
  - Timing Failure Detection

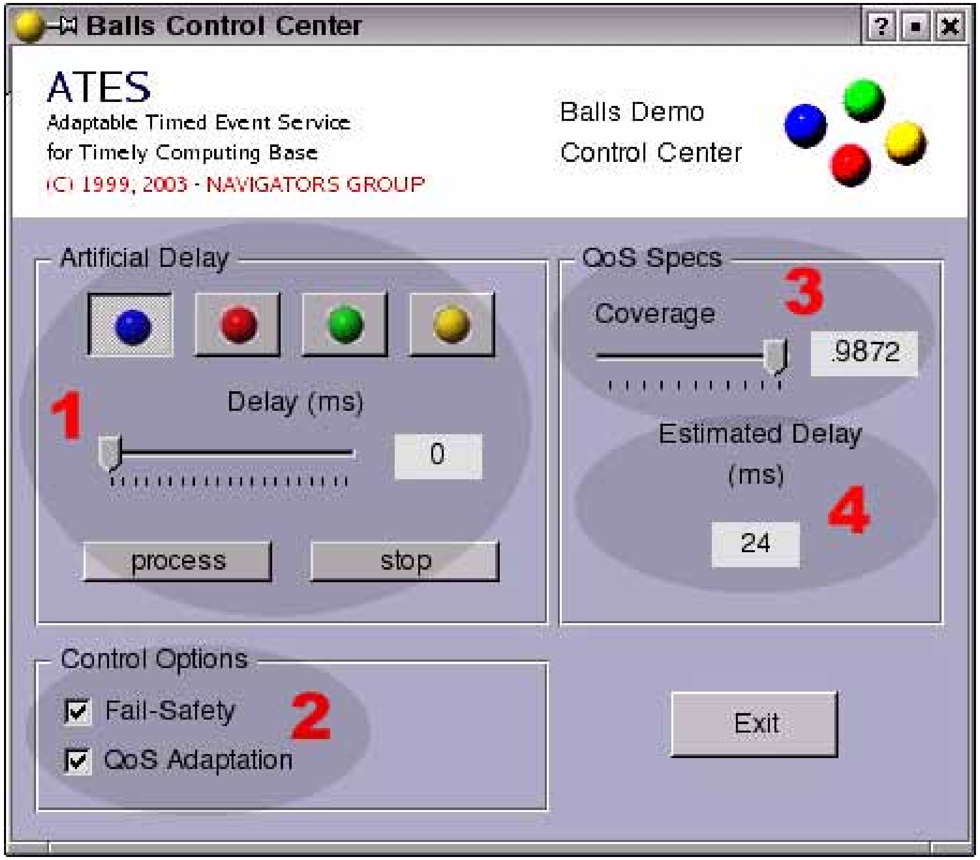
Fail-Safety Demo

- When Fail-Safety is ON:
  - Delivery delay of events is controlled using the TCB distributed TFD
  - Timing failure detected ➔ stop balls in timely way
- When Fail-Safety is OFF:
  - Timing failures can cause balls to crash!

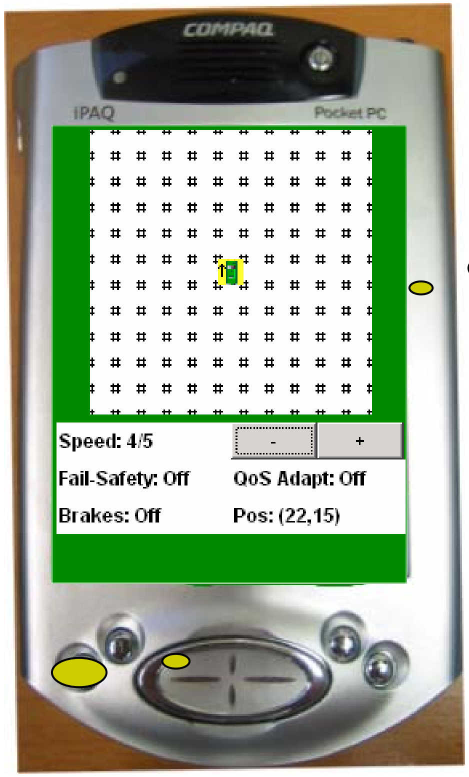
QoS-Adaptation Demo

- When QoS-Adaptation is ON:
  - The service indicates the estimated delay that corresponds to requested coverage value
  - This value is used to determine and set ball speed that preserves safety
  - Coverage stability is achieved
- When QoS-Adaptation is OFF:
  - No speed adaptation takes place
  - Assumed delay keeps constant, possibly leading to coverage degradation due to timing failures

A taste of the experiment…

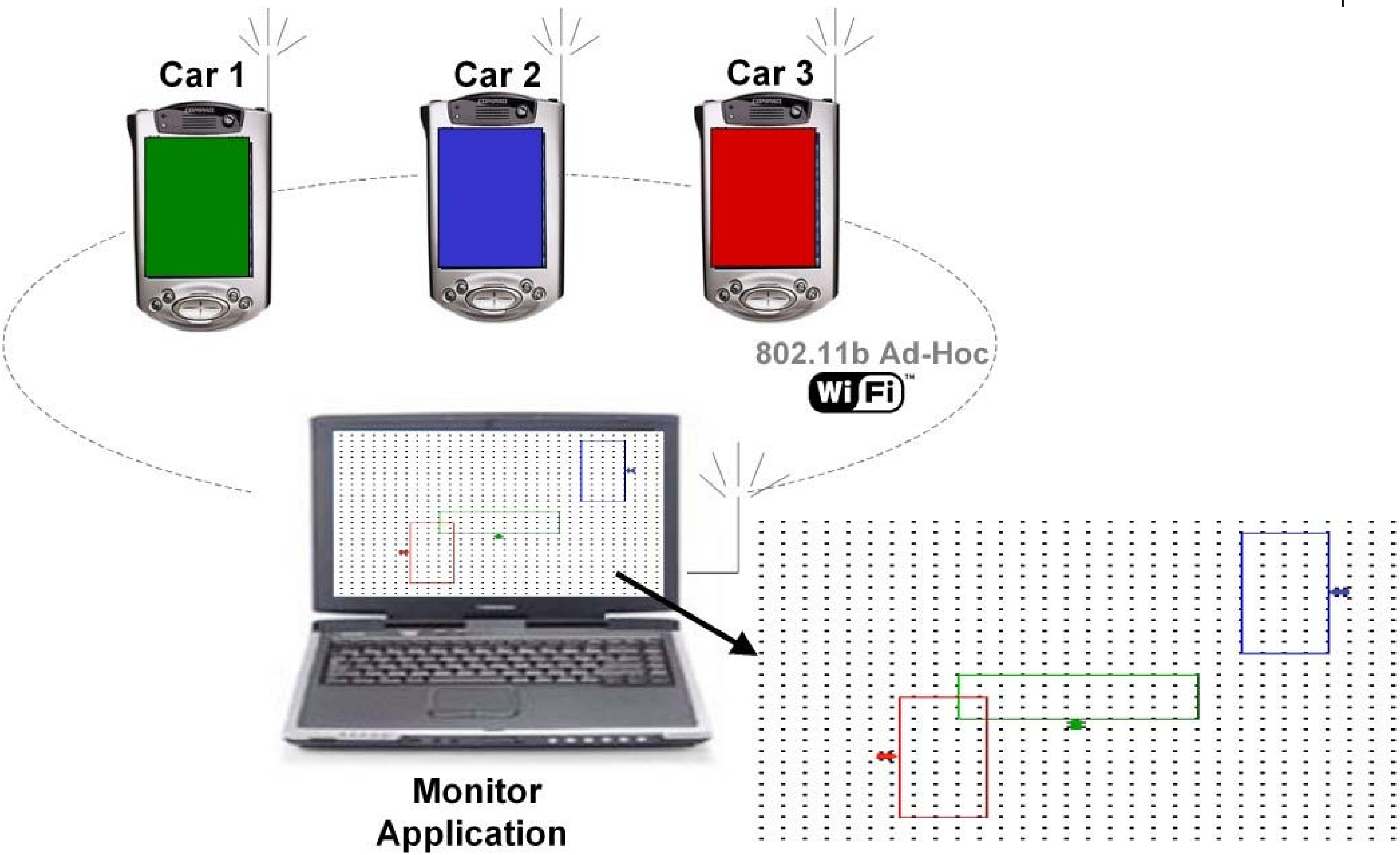


Car (IPAQ) interface



Proximity view

Current state and speed control

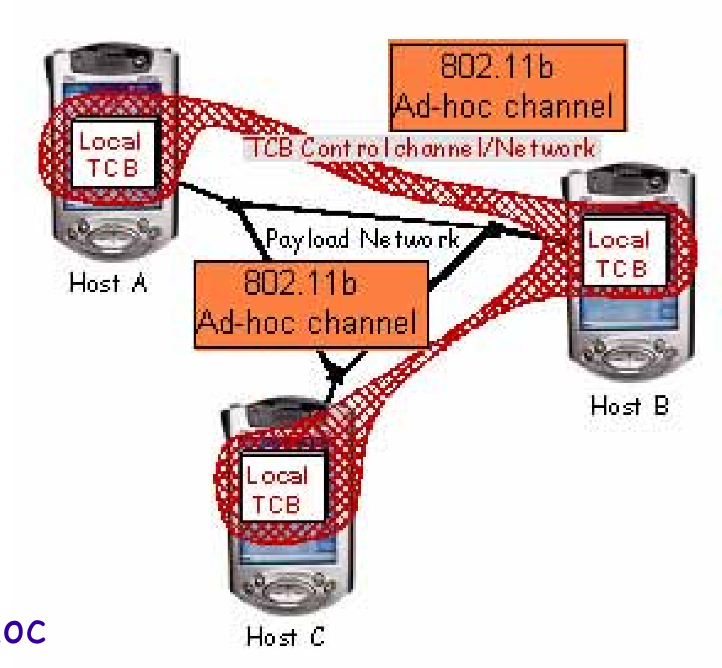

Car 1
Car 2
Car 3
802.11b Ad-Hoc
Wi Fi
Monitor
Application

PARKING LOT!


COMPAQ
iPAQ
Pocket PC
COMPAQ
iPAQ
Pocket PC
COMPAQ
iPAQ
Pocket PC
COMPAQ
COMPAQ
COMPAQ

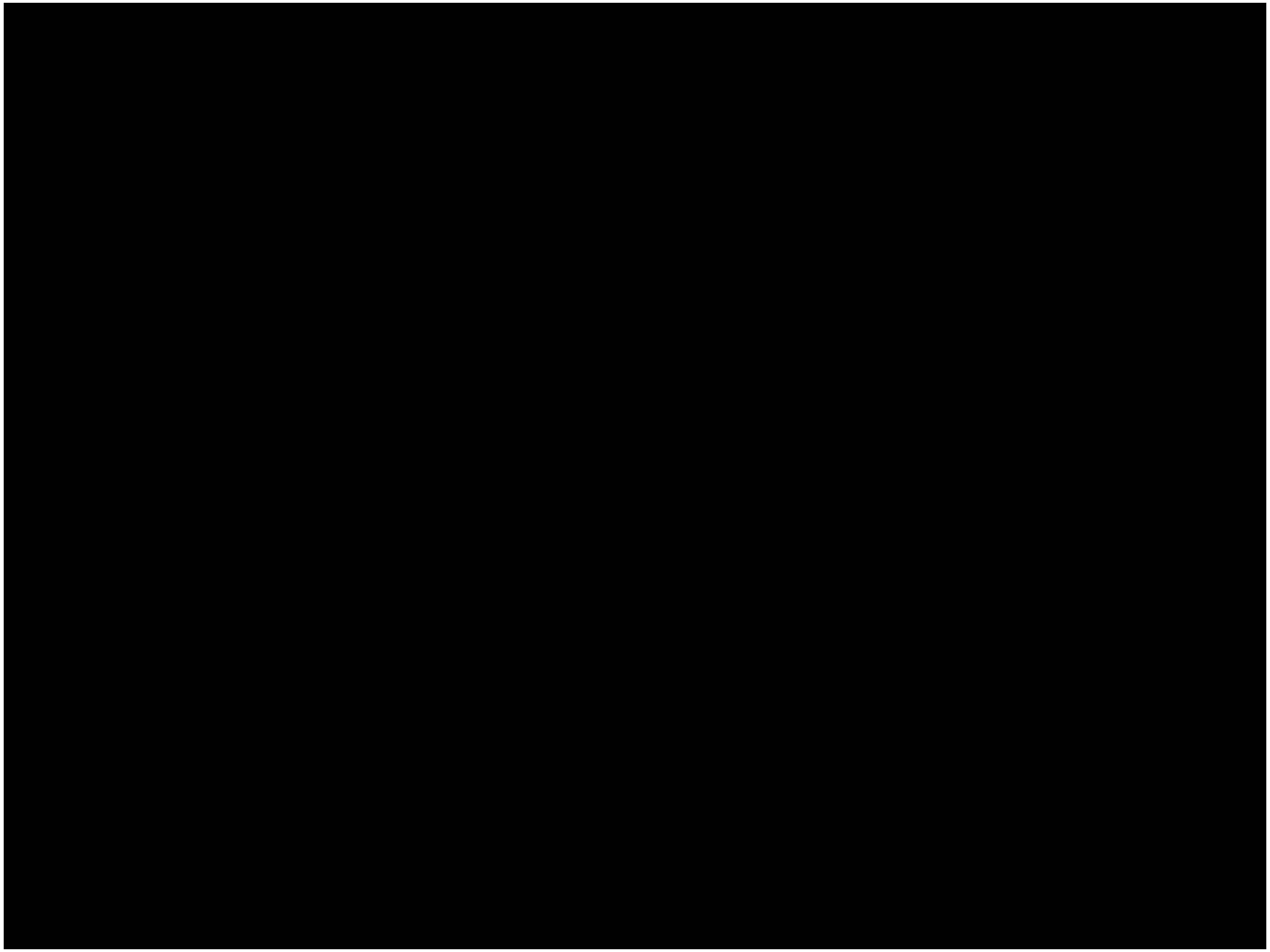
Prototype Implementations

Windows CE / iPAQ pocket PC



- Payload: Windows CE + Regular 802.11b ad-hoc channel
- Windows CE RT tasks + Dedicated 802.11b ad-hoc channel
  - Mockup of a RT protocol (e.g. TBMAC, 802.11e)

Available at http://www.navigators.di.fc.ul.pt/software/tcb



Where is the paper?

- MAIN FEATURE of May 2005 issue of IEEE Distributed Systems On-Line Journal:
  - http://dsonline.computer.org
  - http://dsonline.computer.org/portal/site/dsonline/menuitem.9ed3d9924aeb0dcd82ccc6716bbe36ec/index.jsp?&pName=dso_level1&path=dsonline/0505&file=o5001.xml&xsl=article.xsl&
- *A New Programming Model for Dependable Adaptive Real-Time Applications*
  Pedro Martins, Paulo Sousa, António Casimiro, Paulo Veríssimo
  IEEE Distributed Systems Online, vol. 6, no. 5, 2005.

you may also get there from our web site, www.navigators.di.fc.ul.pt under "Recent Documents“.

Some Recent Publications (w/ urls)

- *On Wormholes and Dependable Adaptation*
- Travelling Through Wormholes: a new look at Distributed Systems Models. **P. Veríssimo**, SIGACTN: SIGACT News, ACM Special Interest Group on Automata and Computability Theory, 37(1), MArch 2006.
- Uncertainty and Predictability: Can they be reconciled? **Paulo Veríssimo**, Future Directions in Distributed Computing, pp. 108-113, Springer Verlag LNCS 2584, May, 2003
- Traveling Through Wormholes: Meeting the Grand Challenge of Distributed Systems. **P. Veríssimo**, Procs. of the International Workshop on Future Directions in Distributed Computing, pages 144-151, Bertinoro-Italy, June 2002.
- *The Timely Computing Base: Timely Actions in the Presence of Uncertain Timeliness*. **Paulo Veríssimo, António Casimiro, C. Fetzer. I**n Proceedings of the 1st International Conference on Dependable Systems and Networks, New York, USA, June 2000.
- The Timely Computing Base Model and Architecture. Paulo Veríssimo, António Casimiro. IEEE Transactions on Computers - Special Issue on Asynchronous Real-Time Systems, vol. 51, n. 8, Aug 2002
- The Timely Computing Base. Paulo Veríssimo and António Casimiro. **Technical Report DI/FCUL TR 99-2, Department of Informatics, University of Lisboa,** May 1999. *(original paper, improved in TOCS02)*

- *Implementing Wormholes*
- *Measuring Distributed Durations with Stable Errors*. **António Casimiro, Pedro Martins, Paulo Veríssimo, Luís Rodrigues.** Proceedings of the 22nd IEEE Real-Time Systs Symposium, London, UK, December 2001
- *How to Build a Timely Computing Base using Real-Time Linux*. **António Casimiro, Pedro Martins, Paulo Veríssimo.** in Proceedings of the 2000 IEEE International Workshop on Factory Communication Systems, Porto, Portugal, September 2000.
- *Timing Failure Detection with a Timely Computing Base*. **António Casimiro, Paulo Veríssimo.** 3rd Europ. Research Seminar on Advances in Distr. Sys (ERSADS'99), Madeira Island, Portugal, April 23-28, 1999
- *The Design of a COTS Real-Time Distributed Security Kernel*, ***Miguel Correia, Paulo Veríssimo, Nuno Ferreira Neves***, *Fourth European Dep. Comp. Conf., Toulouse, France, October 2002 © Springer-Verlag.*

Some Recent Publications (w/ urls)

- *Using Wormholes*
- *Using the Timely Computing Base for Dependable QoS Adaptation*. **António Casimiro, Paulo Veríssimo.** Proceedings of the 20th IEEE Symp. on Reliable Distributed Systems, New Orleans, USA, October 2001
- *Generic Timing Fault Tolerance using a Timely Computing Base*. **António Casimiro, Paulo Veríssimo.** Procs of the Intern'l Conference on Dependable Systems and Networks, Washington D.C., USA, June 2002
- *Efficient Byzantine-Resilient Reliable Multicast on a Hybrid Failure Model*, ***Miguel Correia, Lau Cheuk Lung, Nuno Ferreira Neves, Paulo Veríssimo.*** *Proc's of the 21st Symp. on Reliable Distributed Systems (SRDS'2002), Suita, Japan, October 2002*
- *How to Tolerate Half Less One Byzantine Nodes in Practical Distributed Systems*
  Miguel Correia, Nuno Ferreira Neves, Paulo Veríssimo
  In Proceedings of the 23rd IEEE Symposium on Reliable Distributed Systems. Florianopolis, Brasil, pages 174-183, October 2004
- *Low Complexity Byzantine-Resilient Consensus*
  Miguel Correia, Nuno Ferreira Neves, Paulo Veríssimo, Lau Cheuk Lung
  Distributed Computing, Accepted for publication, 2004. On-line first: http://www.springerlink.com/index/10.1007/s00446-004-0110-7
- *Solving Vector Consensus with a Wormhole*
  Nuno Ferreira Neves, Miguel Correia, Paulo Veríssimo
  Transactions on Parallel and Distributed Systems,2005

Questions?