

# Time-Utility Scheduling and Provably Correct Critical Computer-Based Systems

G rard Le Lann  
INRIA, France

# 1/4: Proof-Based System Engineering for Computer-Based Systems

- Still a challenge: To prove (predictably) that a to-be-fielded X-critical system is correct
- Approach: To distinguish between *system design* issues and *design implementation* issues
- **System design issues are the province of PBSE (Proof-Based System Engineering)**
- Design implementation issues are the province of FSWE (Formal SoftWare Engineering) or FHWE (Formal HardWare Engineering)
- FHWE is mature, FSWE is making progress towards getting mature

- **PBSE is set to catch up with FSWE/FHWE:**
  - PBSE addresses the early lifecycle phases, where most faults are made (under current practice) – contrary to widespread belief, such faults are SE faults, not SWE faults
  - The so-called “SW crisis” cannot be resolved by resorting to SWE techniques/methods, since many problems raised are SE problems
  - The savings potentially achievable with PBSE are gigantic
- **PBSE → system-level proof obligations (POs)**
- Phase 1: Requirement Capture → R = a specification of the application/mission problem to be solved, notably **ADV, the (future) system “adversary”** – all environmental and technological assumptions (e.g., failure & load assumptions)
- Phase 2: System Design & Validation → S = a system specification, along with proofs showing that S implies R
- Implementation/development of S can/should not start until POs are fulfilled (phase 2 is over)

## Excerpts from an IMA problem (military avionics) solved with PBSE (Dassault Aviation, French MoD, INRIA)

### <assumptions>

- process models: graphs, assignment over processors is unrestricted
- processes read/write shared persistent data, any process may read/write any data
- set of processes, set of shared data items, are open-ended
- processor failure models: crash, omission, Byzantine
- process activation models: periodic, sporadic, aperiodic
- computational models: (1) synchronous, (2) pure asynchronous + Chandra & Toueg failure detectors

### <properties>

- safety: process serializability, exactly-once semantics (+ other safety properties “encapsulated” in dependability properties)
- liveness: every process, when activation is requested, eventually terminates
- timeliness: for every process, a strict termination deadline (if no overload)
- timeliness: maximization of a global value function (every process is assigned a constant value) in overload situations
- dependability: consensus, atomic broadcast, group membership, leader election, probability of total system failure  $< 1.10^{-7}$ /hour.

## 2/4: Timeliness Properties & Proofs

- Timeliness (real-time) properties – TimeP – cannot be proved until safety and liveness proofs are established
- TimeP proofs rest on schedulability analyses (worst-case, average case, stochastic, ..)
- With critical systems, worst-case analyses are mandatory, since they are the only way of showing that S will always win against ADV:
  - This cannot be “negotiated” or ignored, since specifying an “adversary” implies that every operational scenario “allowed” as per that specification can/will indeed be instantiated by ADV, worst-case scenarios in particular
  - Necessary-and-sufficient worst-case schedulability analyses are not “pessimistic”, they are no more no less “pessimistic” than decided upon by those who establish the “adversary” specification

- Most often, with “conventional” timeliness attributes (constant deadlines, jitters, ...), proving TimeP raises NP-hard combinatorial problems
  - Certainly “worse” with non “conventional” timeliness attributes, i.e. TU functions
    - proofs of TU-TimeP?
  - Even “worse” when considering:
    - Arbitrary concurrency (e.g., read/write and/or write/write conflicts with accesses to persistent variables shared between distributed processes, possibly multicopied)
    - Partial failures, from crash to omission to Byzantine
    - Distributed systems, i.e. no centralized/unique locus of control (one specific motivation being immunity to partial failures)
- TU-TimeP in Dist. Real-Time & Dependable Systems?**

## 3/4: TU-TimeP & Proofs in DRTD Systems

- With TimeP:
    - Analytical upper bounds on response times –  $B(k)$  for process  $k$  – are established for every process
    - $D(k)$  –  $k$ 's strict termination deadline – is given (specified in  $R$ )
- ▶▶  $\forall k, B(k) < D(k)$
- With TU-TimeP:
    - Analytical lower bounds of achieved utilities –  $U(k)$  for process  $k$  – are established for every process
    - $V(k)$  –  $k$ 's highest utility – is given ( $k$ 's TUF is specified in  $R$ )
    - $0 < \alpha < 1$ ,  $T$  stands for any given time interval,  $K(T)$  = set of processes that may terminate within interval  $T$

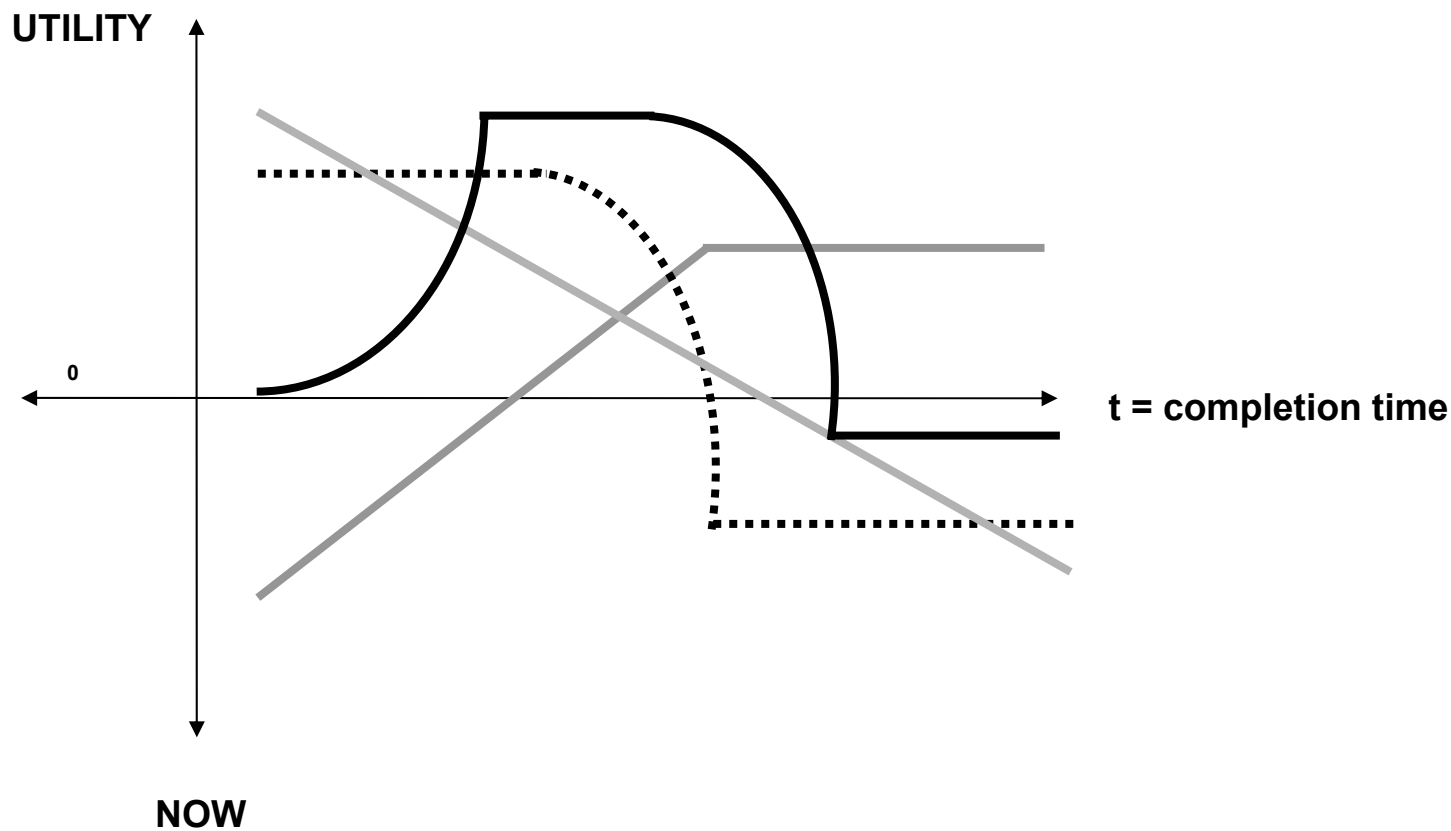
$$\text{▶▶ } \forall T, \forall k \in K(T), \sum_T U(k) / \sum_T V(k) > \alpha$$

- **Schedulers with TU-TimeP:**
  - Specific algorithms, e.g. “best-effort” algorithms designed for “optimizing” aggregate value in the presence of overloads. Typically, complexity  $\approx n^2 \log n$ ,  $n^3$ . No worst-case proofs (no guaranteed  $\alpha$ ) even for nominal cases (no overloads)
  - Algorithms imported from Decision Theory, Game Theory
  - Classical time-driven schedulers (EDF, ...). Typically, complexity  $\approx n \log n$ . Worst-case proofs (guaranteed  $\alpha$ ) exist for nominal cases (no overloads), as well as for overload cases if TUFs are restricted to be constant or univalued functions

- EDF schedulers for non time-varying TUFs:
  - $\forall k$ , pick up bound  $U(k)$ , corresponding to time  $L(k)$ , such that:
    - (1)  $TUF(k) \geq U(k)$  before time  $L(k)$ ,
    - (2)  $TUF(k) \leq U(k)$  after time  $L(k)$
  - De facto,  $L(k)$  is  $k$ 's latest termination deadline for returning a utility at least equal to  $U(k)$ 
    - one can predict a guaranteed  $\alpha$
  - Similarly, one may choose  $E(k)$ ,  $k$ 's earliest termination deadline for returning a utility at least equal to  $U(k)$
  - A particularly important class of TUFs which admit EDF scheduling:

Concave or quasiconcave TUFs

Excerpted from E.D. Jensen's WORDS'03 paper



**Four example time/utility functions**

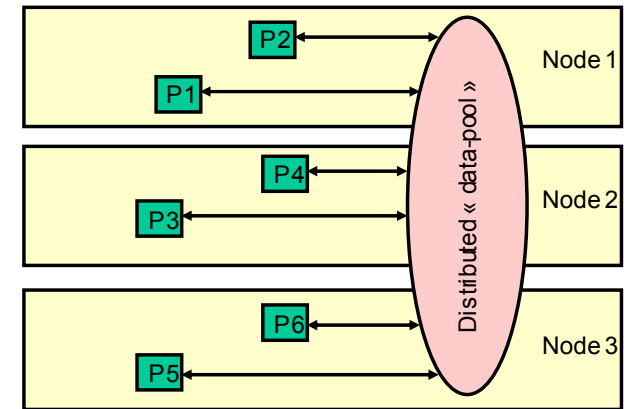
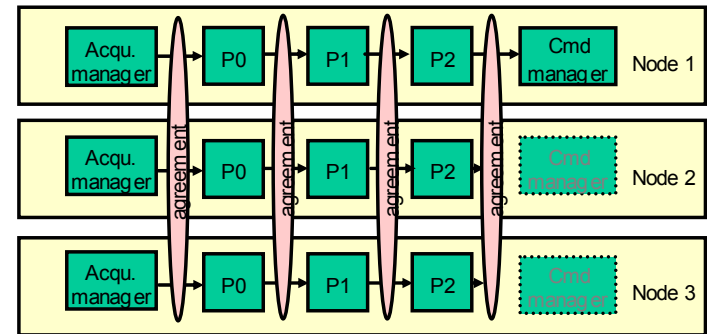
- Overloads?

- Under a PBSE approach, can occur only if the real operational “adversary” is stronger than specified ADV
- Occurrence of an overload is detectable (some process Y is about to miss its deadline L unless its execution is started/resumed immediately – and the CPU is busy with another process Z)
- Various strategies/heuristics can be devised:
  - Short-term scope – e.g. choose to eliminate Y or Z depending on their respective projected returned utilities
  - Broader scope – see past and current work on TUFs

- TUF-driven scheduling in DRTD systems?
  - In order to ensure safety properties – e.g., process atomicity, serializability – and liveness properties, in addition to timeliness properties, it is mandatory to make scheduling decisions out of common knowledge (every scheduler knows that every scheduler knows that ...)
  - For example, schedulers/processors are provided with the same view of the global (system-wide) waiting queue of pending or/and suspended processes/requests,
  - Common knowledge despite concurrency and failures is a class of well known problems in Distributed Computing: Fault-Tolerant Distributed Agreement
    - ➔ Consensus (CS) or Atomic Multi/Broadcast (M/Bcast)
  - There are lots of CS and M/Bcast algorithms, designed and proved in every known computational model, from pure synchrony to pure asynchrony, for every known failure semantics

**Example:** The A3M project (ESA, EADS Astrium, INRIA)

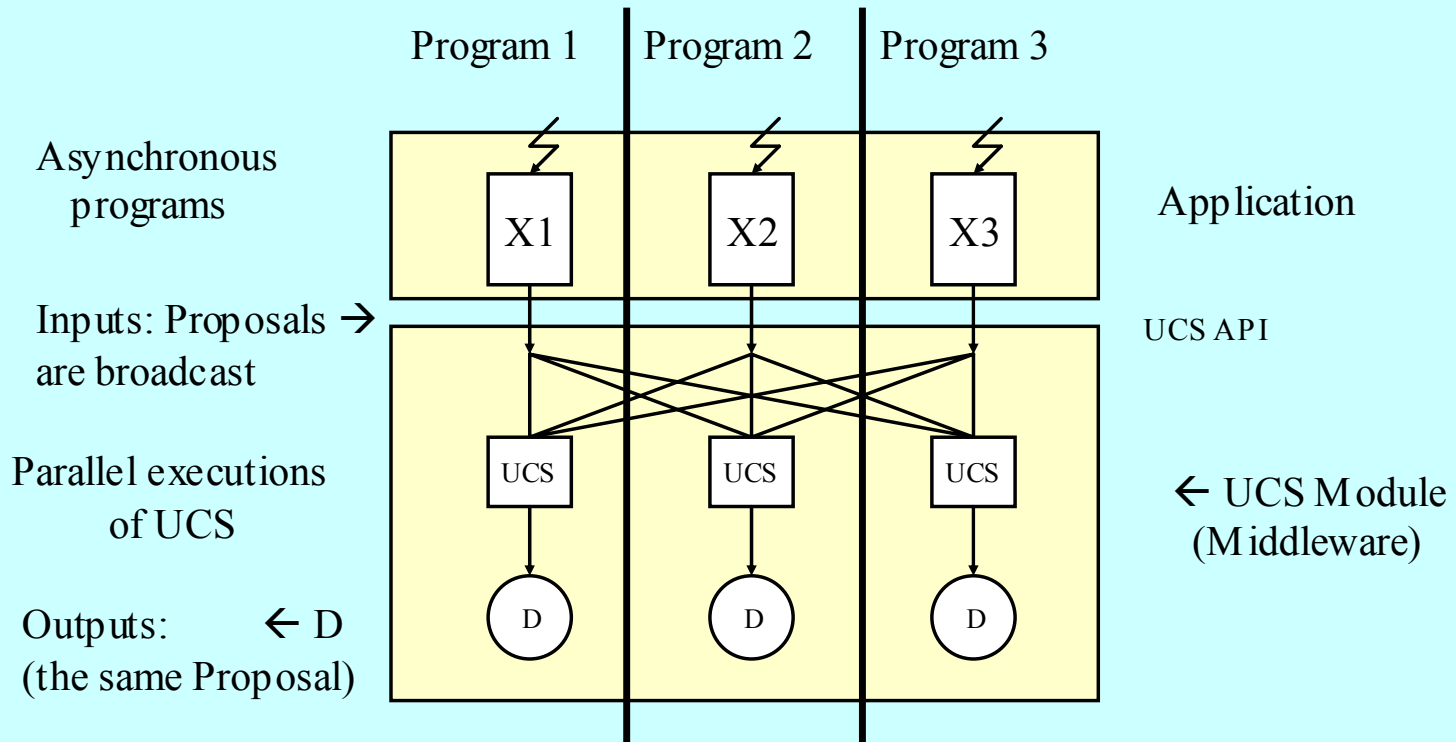
- **Distributed consistent processing under active redundancy**
  - *Some applications require active redundancy with error masking, in order to ensure a very high level of dependability.*
  - *Most solutions implemented today rely on synchronous mechanisms (the computers must be synchronized)*
- **Distributed replicated data consistency, program serialization & program atomicity**
  - *Classical On-Board SW must be able to share/exchange persistent updatable data: the Data Pool*
  - *A reliable design & implementation of a Data Pool is a key to distribution (and mission success)*



## Two algorithms for:

- Uniform Consensus (UCS)
- Uniform Coordination (UCN)

# 3. UCS



# 4. UCN

Programs deposit results of local computations, requests for execution, ... in local buffer B

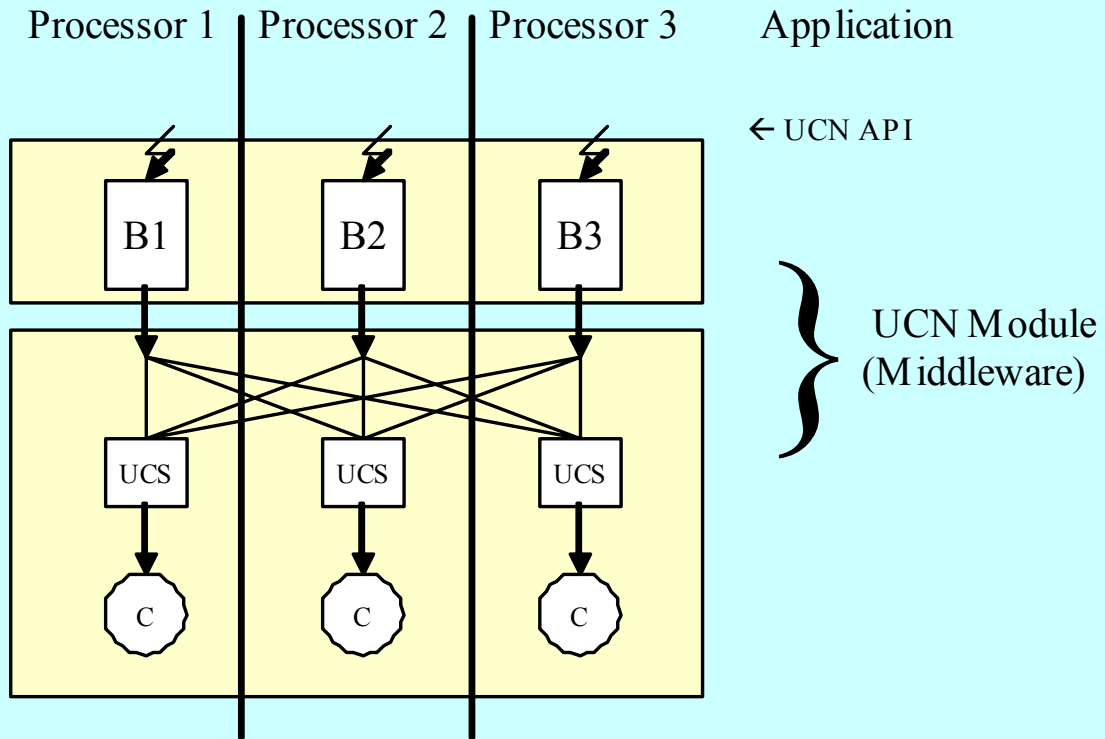
Contents of B (Contributions) are broadcast →

Proposals are computed (function F)

Proposals broadcast →

Parallel executions of UCS

Outputs: ← C (the same Contribution (Global Computation))



- UCN serves to build common knowledge
  - TU-TimeP scheduling is feasible in DRTD systems\*\*
- Coverage?
  - Highest with asynchronous algorithms (UCS, UCN, ...)
- “Timeliness and asynchrony” an oxymoron?
  - Not at all – see the “Design Immersion/Late Binding” principle

**\*\* *UCN is useful also in autonomous systems***

## 4/4: Current & Future Work on PBSE

- INRIA has been pioneering PBSE since 1994 (TRDF)
  - Many contracts have been awarded to INRIA for assessing TRDF/PBSE in practice (Avionics, ATC/ATM, Satellites, Nuclear Power Plants, Telecoms, ...). On our agenda: PBSE tools
- On-going joint work on PBSE (fundamentals, application) with VirginiaTech, VUT, Imperial College, ...
- Two major partners: French MoD, ESA
- Next (major) step – hopefully (2004-2007): The launching of ASSERT
  - An Integrated Project supported by the European Commission, led by ESA (European Space Agency), 30 participating companies
    - ➔ TRDF/PBSE applied & assessed by major space industry companies in Europe
- PBSE felt to be of special relevance to novel unmanned missions (e.g., space), where uncertainty is a certainty!