Self-Adaptive Middleware: A contribution towards taming the complexity of distributed information systems

Yves Caseau

CAL 07 – 3 Octobre

Bouygues Telecom

Overview

- **Part I - Information Systems Complexity**
- Part II – OAI through adaptive middleware
- Part III – Sensitivity to IS patterns
- Part IV – Conclusion

Which complexity ?

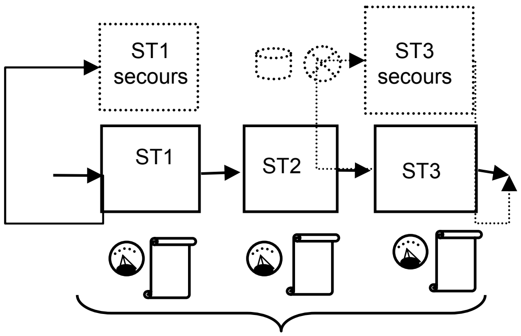
- Size (2005 figures)
  - Over 1 million function points, 50 M TPMC (~1000 servers), 700 To
  - 60% makes a tightly integrated global system (SIC)
  - Impact & Testing makes a larger and larger part of software projects costs
- Time & Dependency
  - Production Planning
  - Project planning
- Quality of Service
  - Customer-facing IT
  - Level of expectation is constantly increasing

Issues resulting from complexity

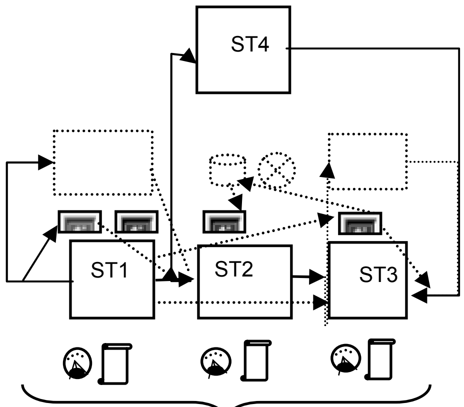
- Quality of Service – meeting business expectation
  - Complexity of heterogeneous process management
- Resilience
  - lowest possible impact of system-scale failures of one or many components
- Coherence of Distributed Data Management & Long Running Transactions
  - ***Practical issue: interaction between signaling flows (process control) and synchronization flows***
  - Assume a separate mechanism that will ensure the coherence of the distributed objects ?
  - Take responsibility of "business object distribution & coherence" as part as the business process management ?
  - Define an acceptable level of "chaos", that is accept that complete coherence is not necessary ?

Biology of Distributed Information Systems

- From a mechanical toward a biology vision of fault-tolerance ☺
  - *Biomimetics meets Information Systems*: when the solutions to the previous issues are emergent properties, not designed.
- An approach that may be applied :
  - System Level : Grid computing, Autonomic computing
  - Process Level: self-adaptive process management (from the infra-structure : topic of this talk)
  - Operations level : « Organic » operations = rely on alternate processes and operations patterns.



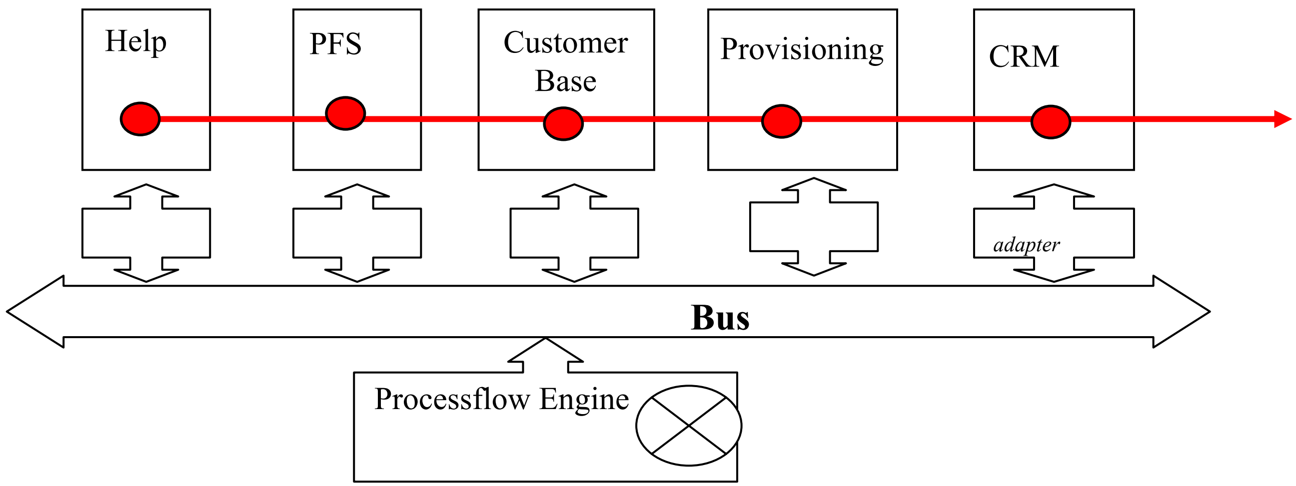
System-based monitoring / recovery



Process monitoring / recovery

OAI : Problem Definition

- Context: (1) business processes which run over a shared set of components



- (2) Service Level Agreements (3) random events

20 clients per Hour in less than 2 minutes

- Activité bursts
- Failures
- Interaction with other processes

- Question: Can process management (load balancing) be automated to maximize business priority satisfaction ?

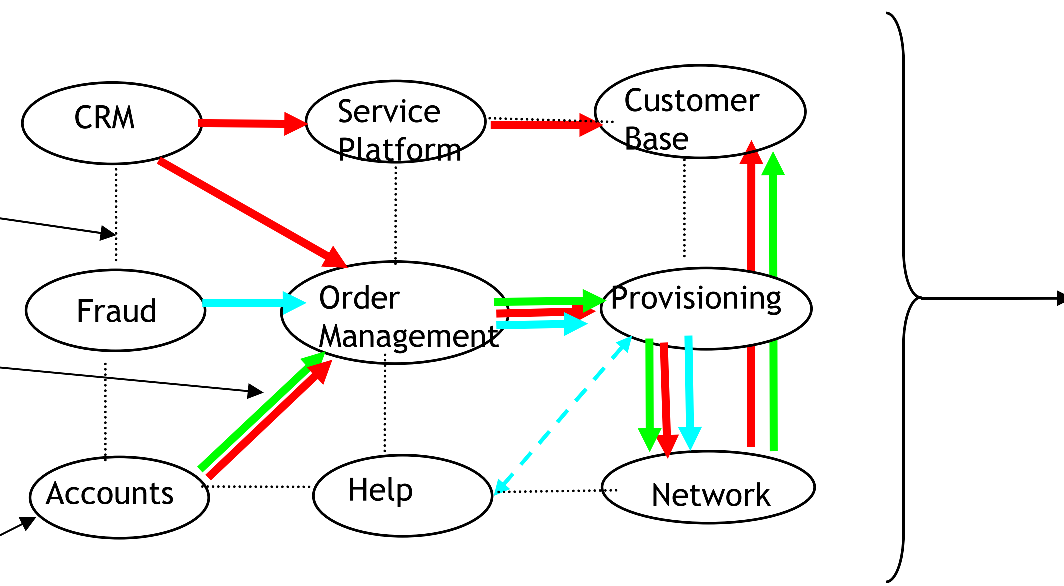
OAI: Optimization of Application Integration

- i-mode™ launch example
  - i-mode subscription is one of many business processes
  - Others include billing / Account management /
  - SLA goals seemed straightforward …

Midleware
-Throughput
-Latency
-Availability
-Message routing

Processes
- SLA
- Priorities

IT Systems
-throughput
-latency
-availability
-Message protocol



Goals (SLA)
- Availability
-Latency
-Throughput
For each process

The challenge of OAI

- Why is OAI hard ?
  - Asynchronous availability is hard to compute
  - Sizing (multi-commodity flow)
  - Stochastic (irregular flows & bursts)
  - Non-linear behavior (message protocol)
  - Monitoring is difficult (for explanations)
  - Functional dependencies between processes (QoS/QoD)
- Culture problem
  - Batch, Client/server, 3/3 architecture have been around for a while -> incident solving know-how
  - Distributed, asynchronous systems that exchange messages are far less common
  - BP culture is long to grow (global perspective)

Overview

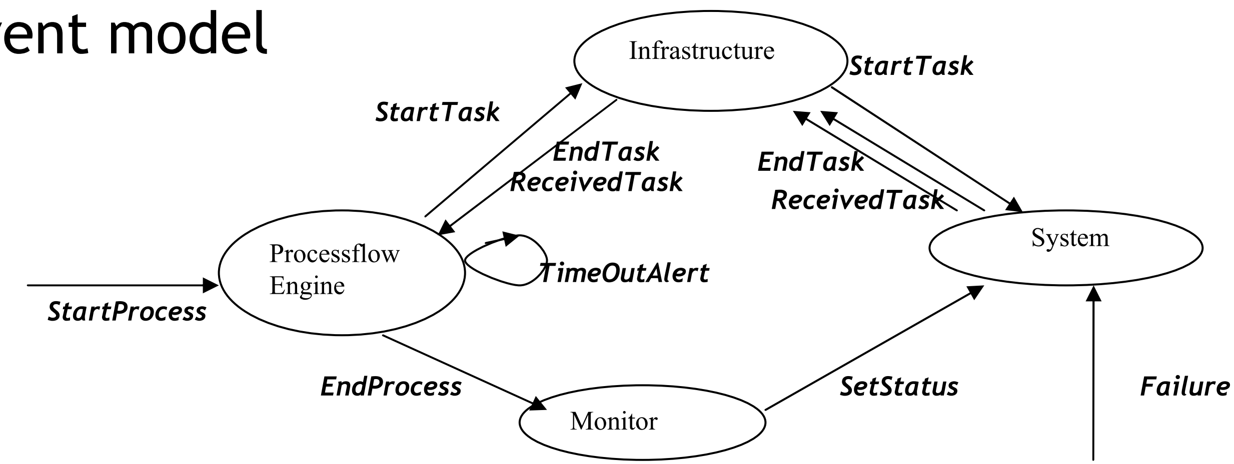
- Part I – Information Systems Complexity
- **Part II - OAI through adaptive middleware**
- Part III – Sensitivity to IS patterns
- Part IV – Conclusion (BDIS)

SLAs, Priorities and Adaptive Strategies

- Each process has a SLA (throughput, latency, availability)
- Business processes have different priorities
  - An adaptive strategy should balance the load according to priorities and SLAs
  - Self-adaptive = tolerance to bursts
  - Self-healing = tolerance to short failures (fail-over)
- Two approaches:
  - Message Handling Rules : modify the order in which messages are handled (higher priority first)
  - Control Rules : slow down lower priority flows

Simulation Model (default)

- 5 Processes (simplified real problem)
  - P1 is a high priority “subscription” process. (high latency)
  - P2 is a medium priority automated baring process.
  - P3 is a lower priority (3) barring.
  - P4 is a high-priority de-barring process (low latency)
  - P5 is a query process of medium priority.
- Finite-event model



- Scenarios to evaluate « graceful degradation »

Routing Strategies

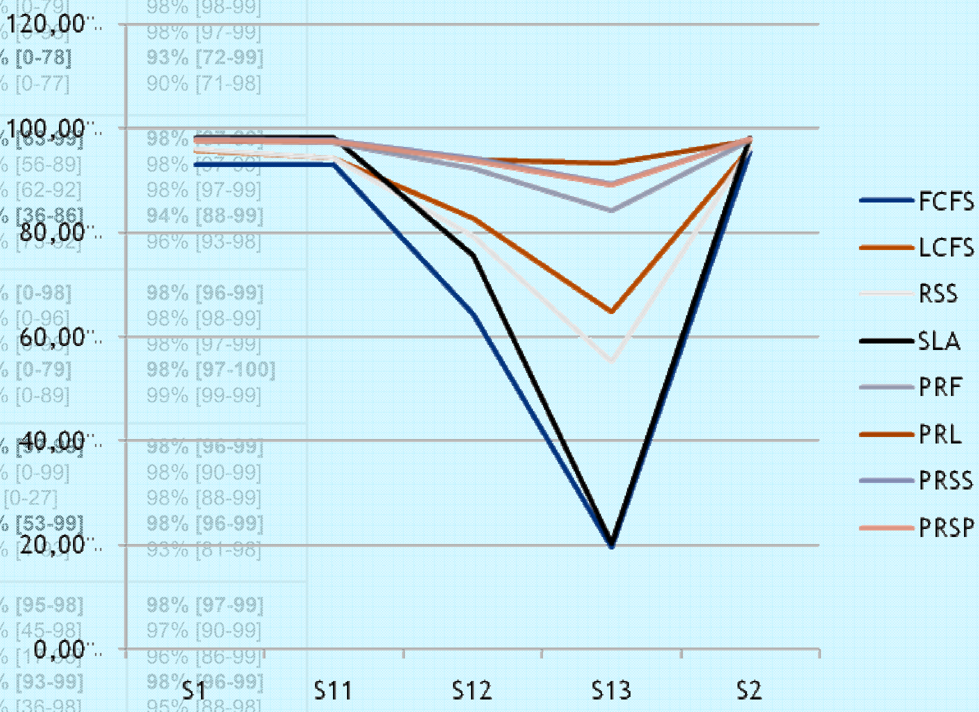
- FCFS (FIFO)
  - Default method for most middleware – respects temporal constraints
  - However, temporal ordering is not preserved by load distribution
- LCFS (FILO)
  - Good strategy for handling backlogs
- "SLA routing"
  - Prediction of processing time based on SLA
  - Sort message according to "expected scheduled time"
- Combination with priorities
  - Process high priority messages first

Scenarios

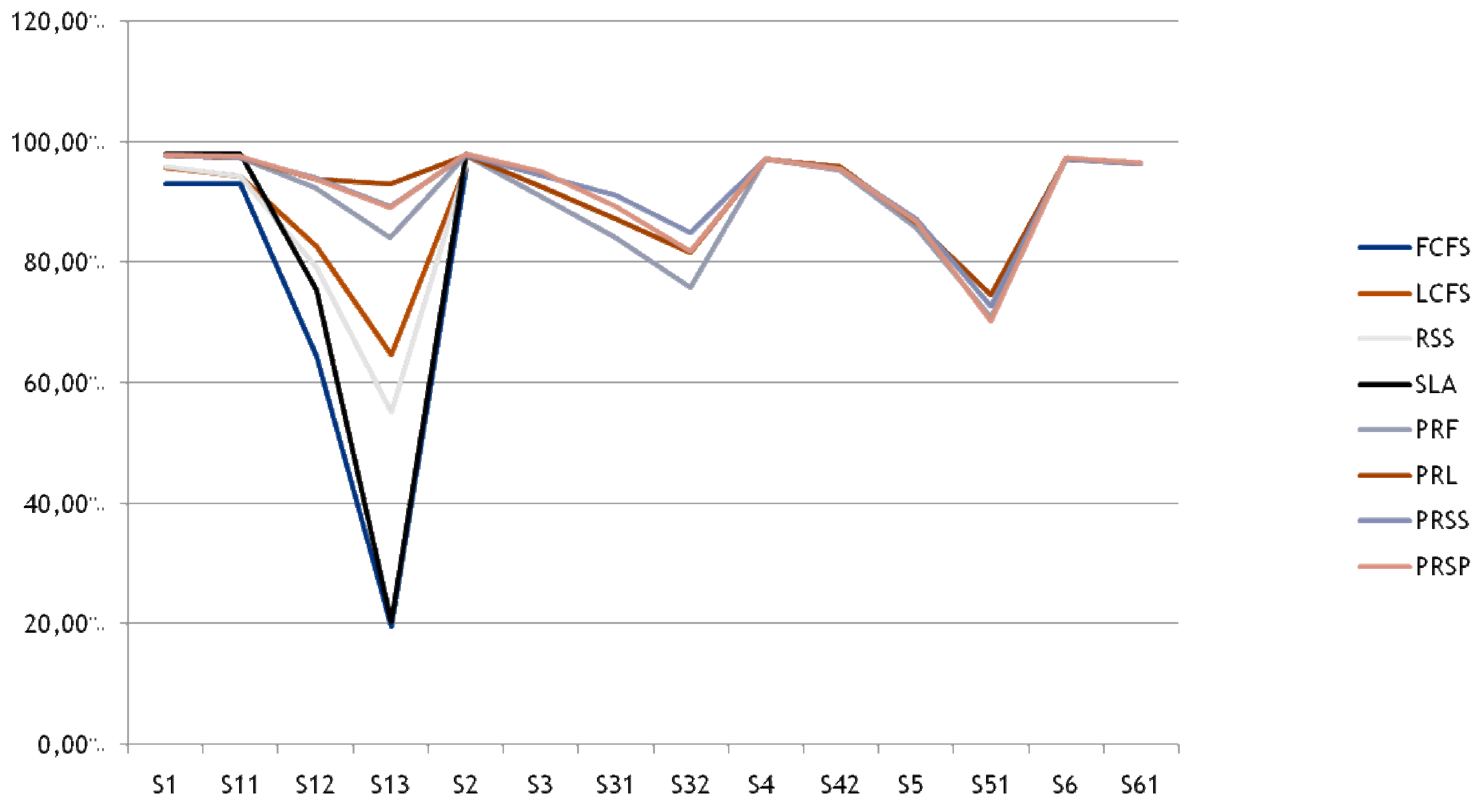
- 3 types of scenarios
  - Reference = static (with overload)
  - Burst (high-priority & low priority)
  - Component failure
- Different event distribution (uniform, Poisson, …)
- Performance evaluation
  - Multiple runs
  - Average, standard deviation of SLA achievement
  - Goal is to observe « graceful degradation » (lower priority processes degrade first)

Computational results (1)

	S1	S11	S12	S13	S2
FCFS	**98% [97-100]** 98% [98-99] 98% [97-99] **88% [76-98]** 84% [71-99]	**98% [97-100]** 98% [98-99] 98% [97-99] **88% [76-98]** 84% [71-99]	**83% [45-99]** 59% [4-99] 76% [38-99] **47% [0-98]** 46% [0-99]	**28% [0-98]** 15% [0-79] 22% [illegible] **12% [0-78]** 13% [0-77]	**98% [97-99]** 98% [98-99] 98% [97-99] **93% [72-99]** 90% [71-98]
LCFS	**98% [97-99]** 98% [96-99] 98% [97-99] **93% [86-97]** 96% [94-98]	**98% [97-99]** 96% [93-98] 98% [96-99] **90% [81-97]** 94% [90-99]	**92% [77-99]** 85% [71-99] 91% [79-99] **72% [50-97]** 86% [76-99]	**75% [63-99]** 66% [56-89] 73% [62-92] **52% [36-86]** 79% [73-92]	**98% [97-99]** 98% [97-99] 98% [97-99] **94% [88-99]** 96% [93-98]
SLA	**98% [97-99]** 98% [98-99] 98% [97-99] **98% [96-99]** 99% [99-99]	**98% [97-99]** 98% [98-99] 98% [97-99] **98% [96-100]** 99% [98-99]	**82% [47-99]** 75% [39-99] 78% [39-99] **69% [25-99]** 74% [39-99]	**26% [0-98]** 18% [0-96] 22% [illegible] **15% [0-79]** 20% [0-89]	**98% [96-99]** 98% [98-99] 98% [97-99] **98% [97-100]** 99% [99-99]
PRF	**98% [97-99]** 97% [91-99] 97% [84-99] **98% [94-100]** 93% [88-99]	**98% [96-99]** 96% [86-99] 94% [80-99] **98% [93-99]** 89% [80-99]	**98% [97-99]** 81% [45-99] 44% [0-99] **95% [85-99]** 69% [24-99]	**98% [97-99]** 48% [0-99] 2% [0-27] **86% [53-99]** 45% [illegible]	**98% [96-99]** 98% [90-99] 98% [88-99] **98% [96-99]** 93% [81-98]
PRL	**98% [96-99]** 97% [92-99] 97% [87-99] **98% [95-100]** 95% [91-98]	**98% [96-99]** 96% [92-99] 92% [82-99] **98% [95-99]** 93% [86-99]	**98% [97-99]** 83% [54-99] 54% [7-99] **96% [88-99]** 82% [63-99]	**97% [95-98]** 73% [45-98] 59% [illegible] **97% [93-99]** 70% [36-98]	**98% [97-99]** 97% [90-99] 96% [86-99] **98% [96-99]** 95% [88-98]
PRSS	**98% [97-100]** 97% [91-99] 97% [91-99] **98% [96-99]** 96% [91-99]	**98% [97-99]** 96% [88-99] 94% [80-99] **98% [97-99]** 94% [86-99]	**98% [96-99]** 83% [51-99] 44% [0-99] **98% [97-99]** 79% [49-99]	**98% [96-99]** 50% [0-99] 3% [0-30] **97% [91-99]** 52% [3-97]	**98% [97-100]** 98% [91-99] 98% [96-99] **98% [97-99]** 96% [87-99]



Computational Results (II)



Results

- Priority routing works. The algorithms that use process priority as part of the sorting strategy are able to maintain the SLA of high priority processes much longer.

- The second lesson is that FCFS is not a good default algorithm. LCFS does better as soon as the event flow become tight.

- The combination of priority and SLA sorting is the best approach.

Flow Rules

- First intuition at Bouygues Telecom was to implement control flow mechanisms (emergency mode)
- Before actually implementing it in the EAI adapter, we use the simulation engine to evaluate two strategies :
  1. RS1: When the QoS of a system X fails lower than 90% of its SLA level (cf. Section 3), we reduce the flow of systems that are providers of X whose priority is lower than X. A dual rule restores the default setting once the QoS of X reaches 90%.
  2. RS2: This is a similar rule, but the triggering condition is based on processes. When the QoS of a process P fails below 90%, we reduce the flow of all systems that have a lower priority than P and who are providers of a system that supports P.
- Control flow is more complex to operate but it is not necessarily part of the middleware infrastructure

Routing Rules

- We implemented rules that dynamically change the message handling strategy (using a “status” : FAST means use PRL to process a backlog)
  - RS3: When the QoS of a system X drops below 95%, the system is switched to FAST status. The system resumes normal status once the QoS returns above 95%.
  - RS4: When the QoS of a process P drops below 95%, all systems that support this process are switched to FAST status.
  - RS5: A system is switched to FAST status whenever its mailbox size grows over 100. Obviously, the triggering size is a constant that depends on the volume that is processed by the EAI and the number of connected systems.

Results

	S33		S51		S2	
	FCFS	PRSP	FCFS	PRSP	FCFS	PRSP
No Rules	**38%** [illegible]	**70%** [illegible]	**56%** [illegible]	**75%** [illegible]	**98%** [illegible]	**98%** [illegible]
RS1	[illegible] 42% **33%** 31%	[illegible] 23% **65%** 39%	[illegible] 55% **47%** 35%	[illegible] 33% **70%** 52%	[illegible] 98% **93%** 81%	[illegible] % 98% **98%** 92%
RS2	**52%** 25% 46% **25%** 33%	**70%** 43% 23% **65%** 66%	**62%** 29% 55% **46%** 35%	**75%** 61% 33% **70%** 66%	**98%** 98% 98% **93%** 90%	**98%** 98% 98% **98%** 97%

Does not provide any stable improvement

	S33		S51		S2	
	PRF	PRSP	PRF	PRSP	PRF	PRSP
No Rules	**69%** 42% 23% **63%** 65%	**70%** 44% 22% **66%** 67%	**76%** 62% 37% **70%** 72%	**75%** 61% 33% **70%** 75%	**98%** 98% 98% **98%** 93%	**98%** 98% 98% **98%** 97%
RS3	**74%** 69% 58% **75%** 72%	**75%** 69% 59% **77%** 72%	**76%** 69% 65% **73%** 79%	**74%** 68% 64% **72%** 80%	**98%** 97% 98% **98%** 92%	**98%** 98% 98% **98%** 96%
RS4	**71%** 64% 52% **69%** 67%	**76%** 68% 57% **74%** 70%	**76%** 66% 59% **72%** 78%	**74%** 64% 59% **69%** 78%	**98%** 98% 98% **98%** 93%	**98%** 98% 98% **98%** 97%
RS5	**77%** 74% 65% **77%** 72%	**78%** 73% 63% **80%** 74%	**77%** 74% 65% **77%** 72%	**75%** 66% 57% **72%** 80%	**98%** 98% 98% **98%** 93%	**98%** 98% 98% **98%** 97%

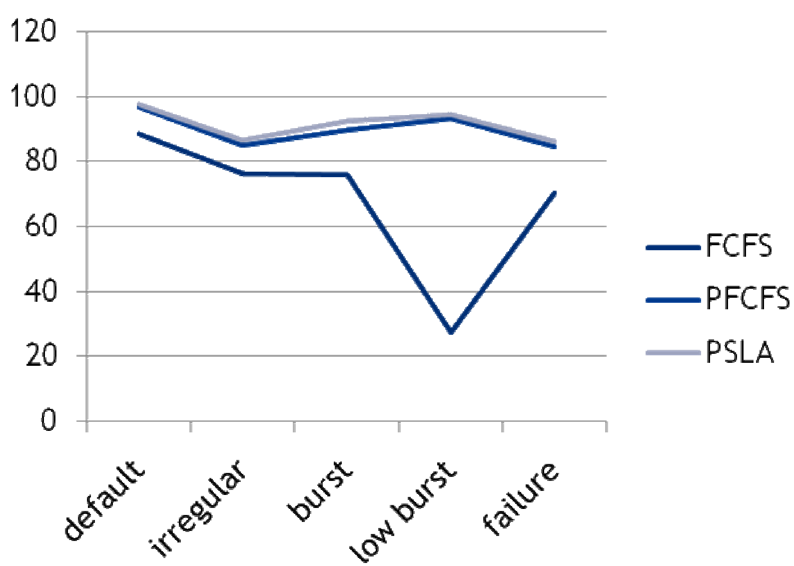
-Small improvement
-Simpler is better

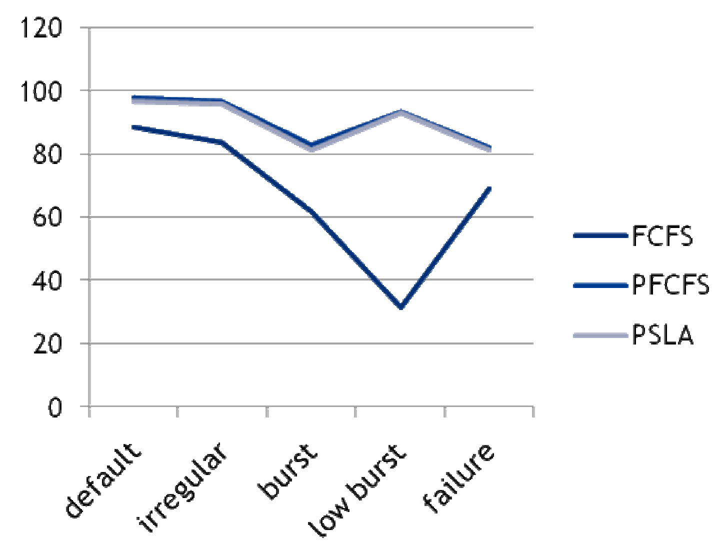
Overview

- Part I – Information Systems Complexity
- Part II – OAI through adaptive middleware
- **Part III - Sensitivity to IS patterns**
- Part IV – Conclusion (BDIS)

IS patterns (I) : short services

- Default vs. Short processes

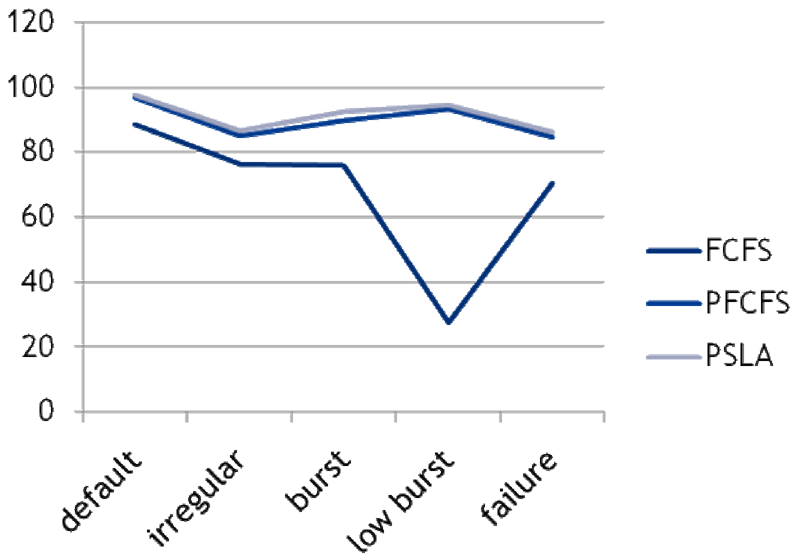


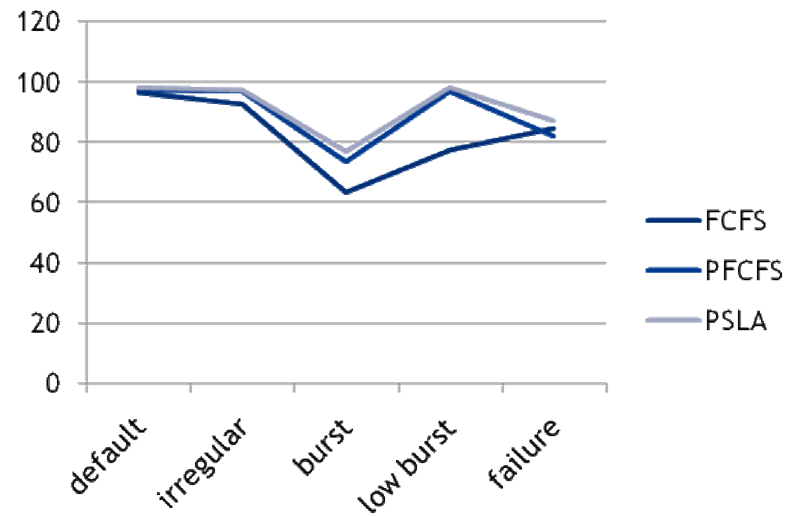


- Irregular load is easier to manage with shorter processes
- The opposite is observed with bursts

IS patterns (II) : longer processes

- default vs longer processes (non-homogeneous)

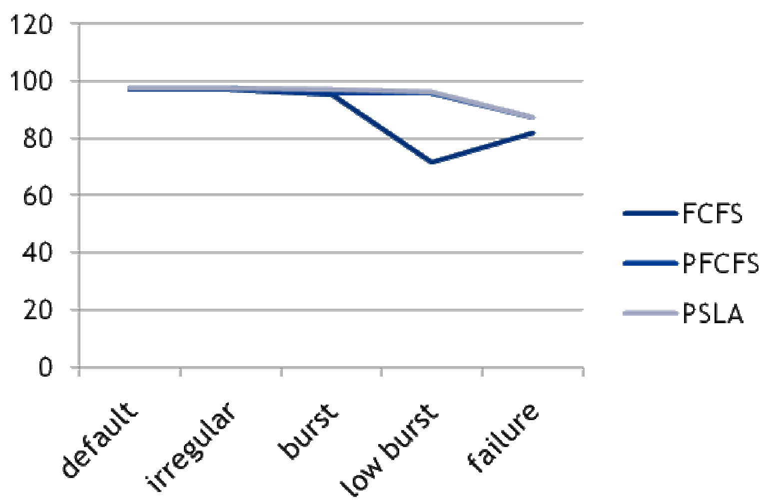


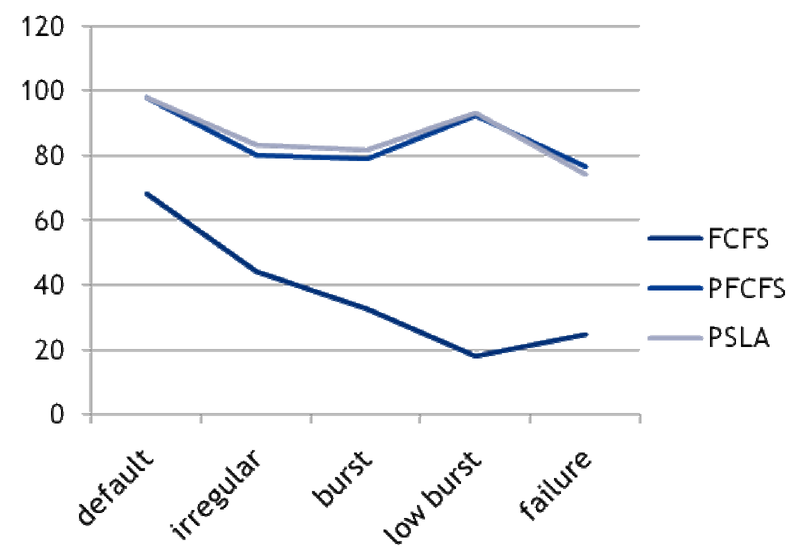


- different difficulty patterns …
- … but the relative ranking of methods is not changed

IS patterns (III) : « *Lean Manufacturing* »

- Contrast
  - « lean » IS : 60% capacity usage, tight SLA (50% lean ratio)
  - « optimized IS »: 80% capacity usage, loose SLA (10-20% lean ratio)





- An experimental verification of Taichi Ohno's intuition (Toyota) ☺ lean = under-optimization of resources to achieve flexibility and robustness.

Conclusions

A first step towards "autonomic BPM"

1. Self-optimization:
   - Priority handling works: it is possible and fairly simple to take process priority into account for routing messages and the results show a real improvement.
   - Routing (mailbox sorting) algorithm matters: the more sophisticated SLA projection technique showed a real improvement over a FCFS policy.
   - Control rules are interesting, but they are secondary to the routing policy: it is more efficient to deal with congestion problems with a distributed routing strategy rather than with a global rule schema.
2. Self-healing: some form of self-healing is demonstrated but true self-healing requires collaboration with HW
3. Self-configuration: the goal is to make configuration declarative (e.g., SLA) vs. defining time & resource configuration (e.g., schedules)