Probabilistic QoS soft contract for Web Service orchestrations

A. Benveniste, S. Rosario, S. Haar, C. Jard

IRISA, Rennes

Outline

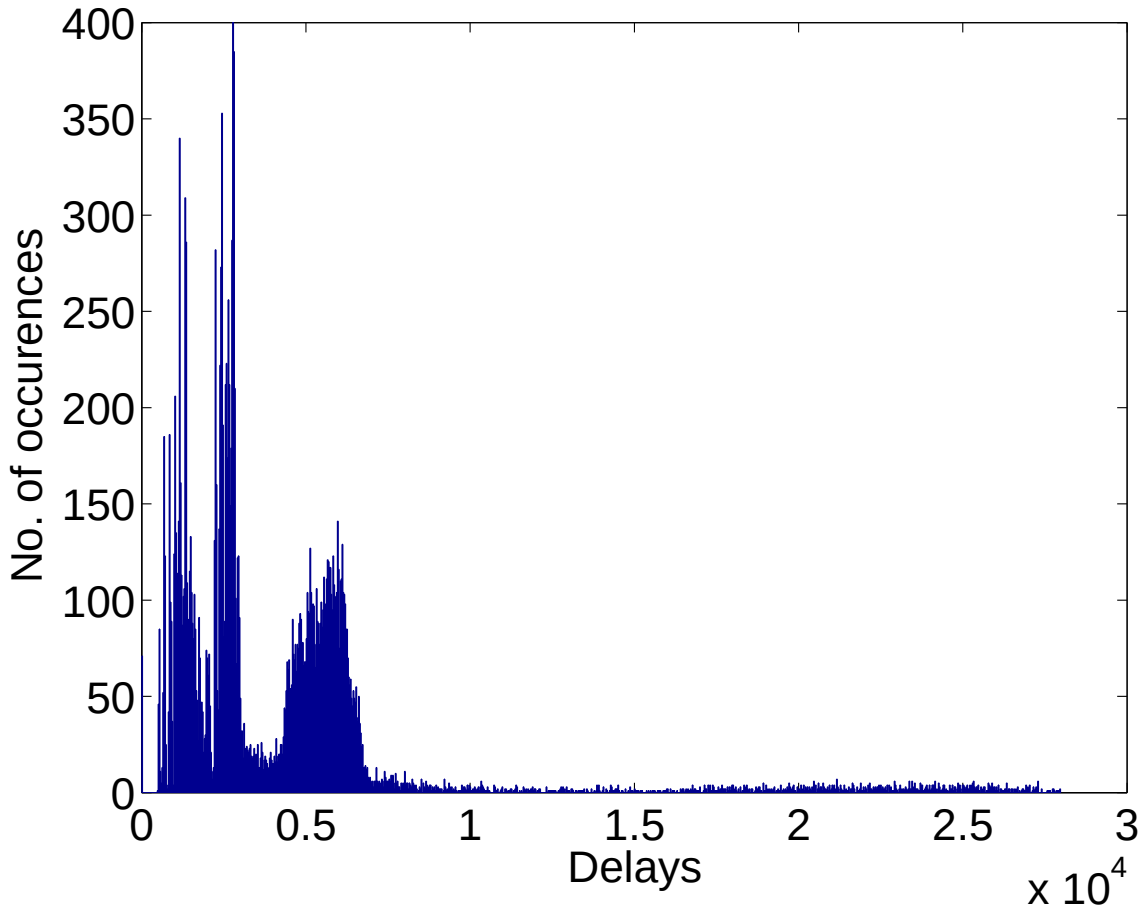
Motivations

- Web Service orchestrations : composition of Web services.
- Orchestrations establish contracts
  1. between orchestration and user, and
  2. between orchestration and called services.
- Contract Composition: deriving 1 from 2.
- Aim: Develop formal tools to perform contract composition
- Contract : Agreed constraints on functional and QoS parameters (response time, throughput, availability, etc.)

Hard/Soft Contracts

- Contract composition is typically performed based on *hard* contracts:
  - Response time always less than 5 milli seconds
- May not fit with the real behaviour of Web services, however...

Hard/Soft Contracts



Response times of 20,000 calls to a Stock Quote Service.

Hard/Soft Contracts

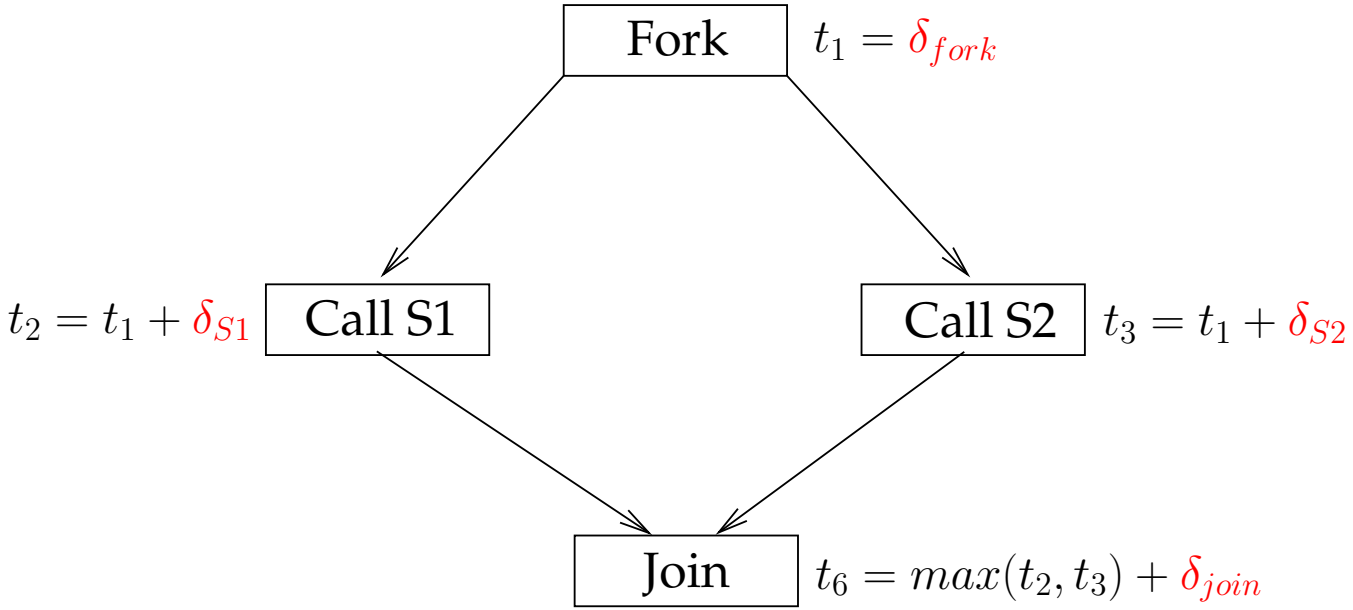
- Alternative: use *soft* contracts:
  - Response time less than 5 milli seconds in 95% cases

  Unfortunately, *soft contracts cannot be composed* — this is why they are not used in contract composition

- Solution: use **probabilistic soft** contracts:
  - For QoS parameter "Response Time", I am offering this probability distribution, and I promise to behave no worse
- *Probabilistic soft contracts can be composed*
- Simulations show significant room for safe *overbooking*

Outline

Soft Probabilistic Contract Composition



- sample δ's at random from contract distributions
- for each sample, evaluate latency using max+ rules
- yields samples of t_6 = latency, for the orchestration
- estimate orchestration contract from these samples

TOrQuE **T**ool for **Or**chestration simulation and **Qu**ality of service **E**valuation

The *TOrQuE* tool implements our approach:

- *inputs*:
  - contracts with called Web services
  - functional model of the orchestration
- *outputs*:
  - empirical distributions of QoS parameters for the orchestration

TOrQuE: Functional Model of the Orchestration using Orc

- Orc: Formal model for distributed computation (Misra et al. U Texas at Austin)

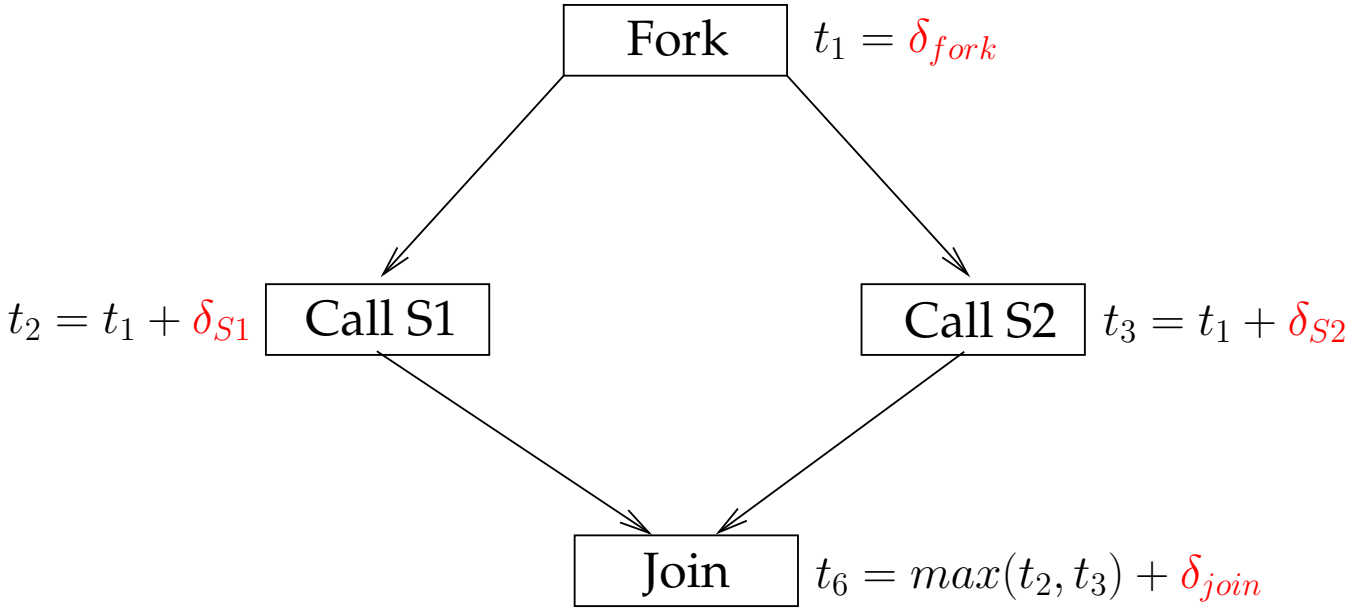
- Basic computation element: Site.
- Operators to compose site calls into complex Orc expressions.
- Allows definition of recursive expressions

- Simpler and cleaner than BPEL, yet it provides the essential features needed for orchestrations

TOrQuE: Time Stamper

- Computes the delays in performing actions of the orchestration
- For each action, this delay could be produced in different ways:
  - Random Generation The delay is sampled from a distribution specified in the contract.
  - BootStrapping In case no contract is available, we could also perform bootstrapping on a set of pre-measured delay values. For *e.g.*, record a certain number of delay values from calls to the site and uniformly sample from them.

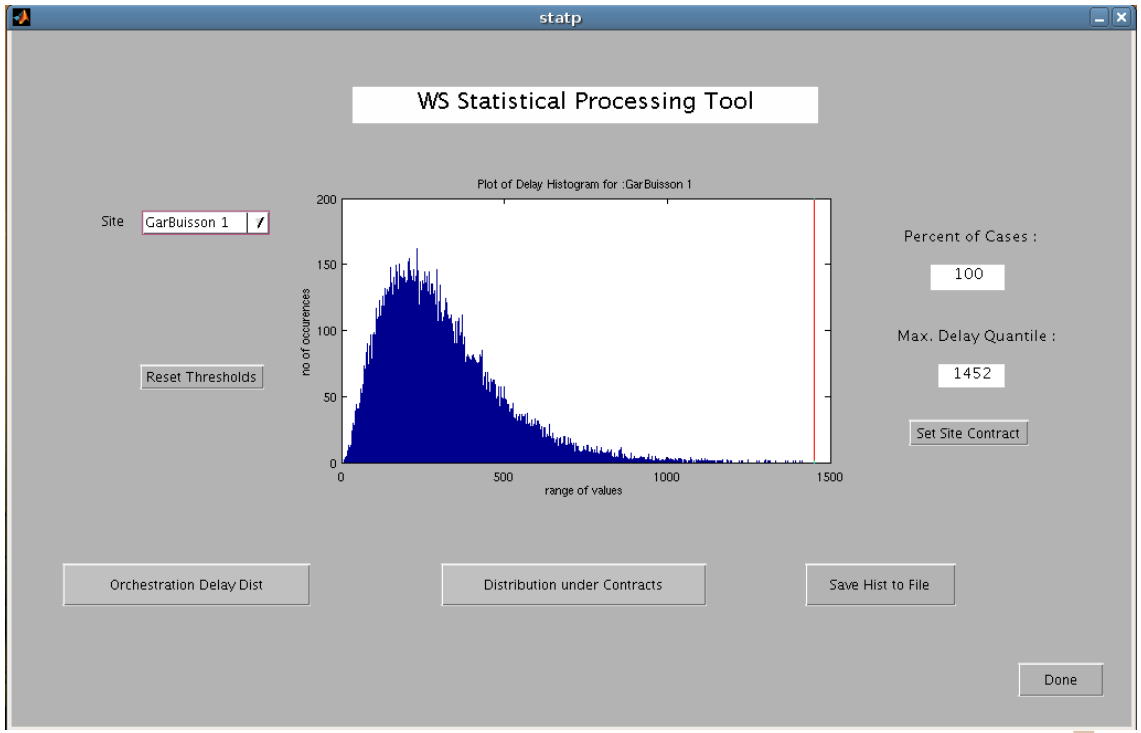
TOrQuE: Trace Reconstructor



- For complex orchestrations, the runs, seen as partially ordered sets of events, are dynamically defined.
- Orc $\longrightarrow$ Event Structure, collecting all such possible runs (joint work with the authors of Orc)

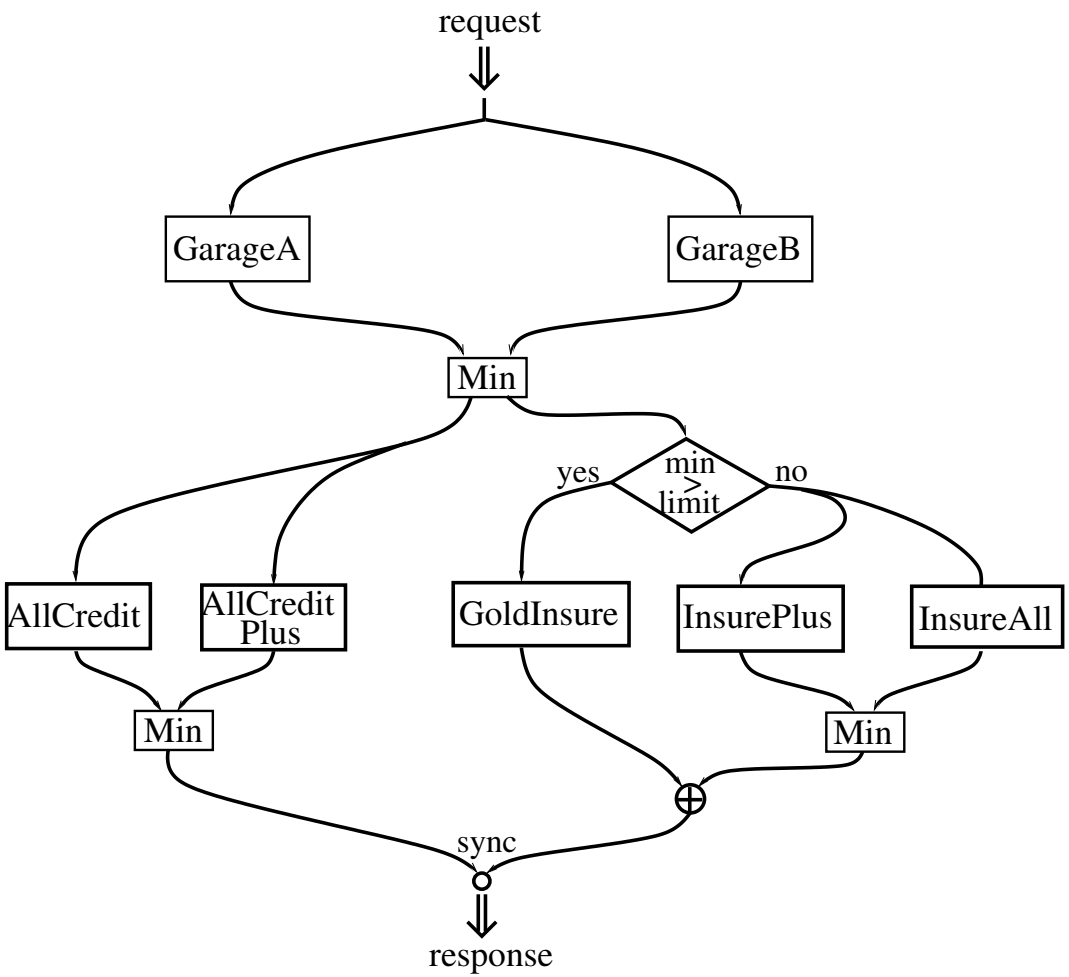
TOrQuE: SLA Design

- Mainly a GUI module implemented in MATLAB
- Allows visualizing empirical distribution of individual sites and the overall orchestration, and then selecting the actual quantile to establish an "ordinary" soft contract



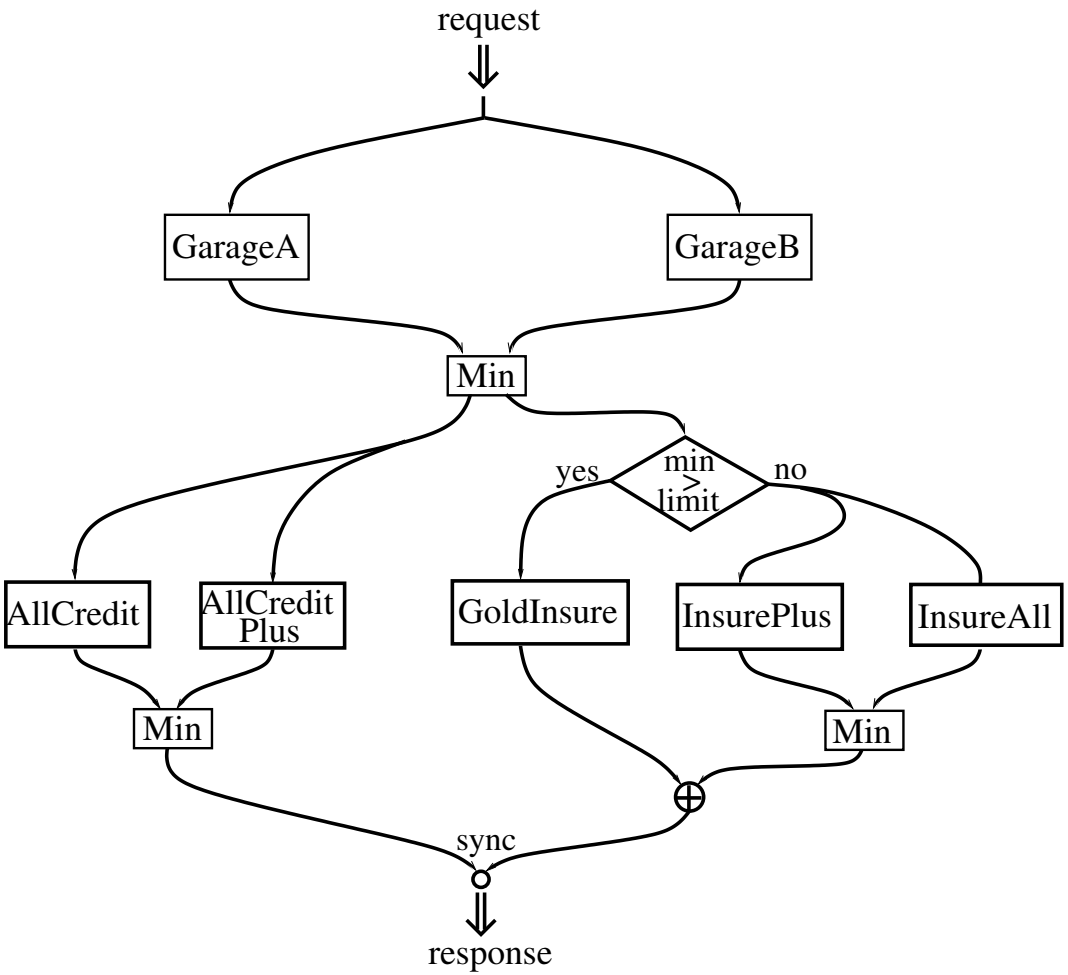
Outline

Orchestration Example : CarOnLine



1. Get quotes for car price from a list of garages
2. Parallel calls to garages guarded by timeouts.
3. Get credit and insurance proposals for best offered car price.

CarOnLine Example - Schematic figure and Orc program



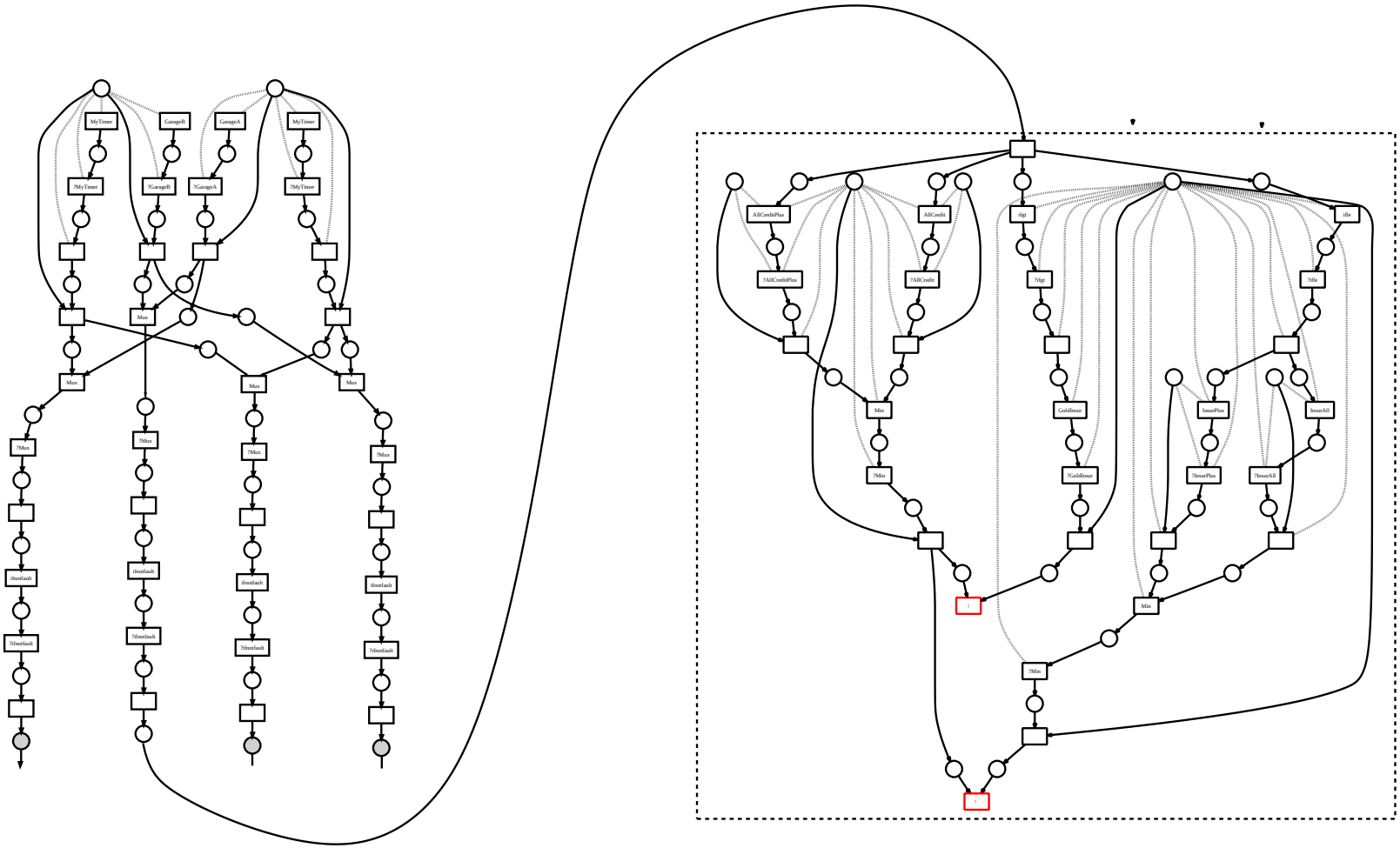
$$CarOnline(car) \triangleq CarPrice(car) \gg p \gg let(p, c, r) \textbf{ where } c :\in GetCredit(p),\ r :\in GetInsur(p)$$

$$CarPrice(car) \triangleq \{Mux(p1, p2) \textbf{ where } p1 :\in GarageA(car) \mid Timer(T),\ p2 :\in GarageB(car) \mid Timer(T)\} \gg p \gg if(p \neq Fault)) \ggg let(p)$$

$$GetCredit(p) \triangleq Min(r1, r2) \textbf{ where } r1 :\in AllCredit(p),\ r2 :\in AllCreditPlus(p)$$

$$GetInsur(p) \triangleq \{if(p \geq limit) \ggg GoldInsure(p)\} \mid \{if(p \leq limit) \ggg min(ip, ia) \textbf{ where } ip :\in InsurePlus(p),\ ia :\in InsureAll(p)\}$$

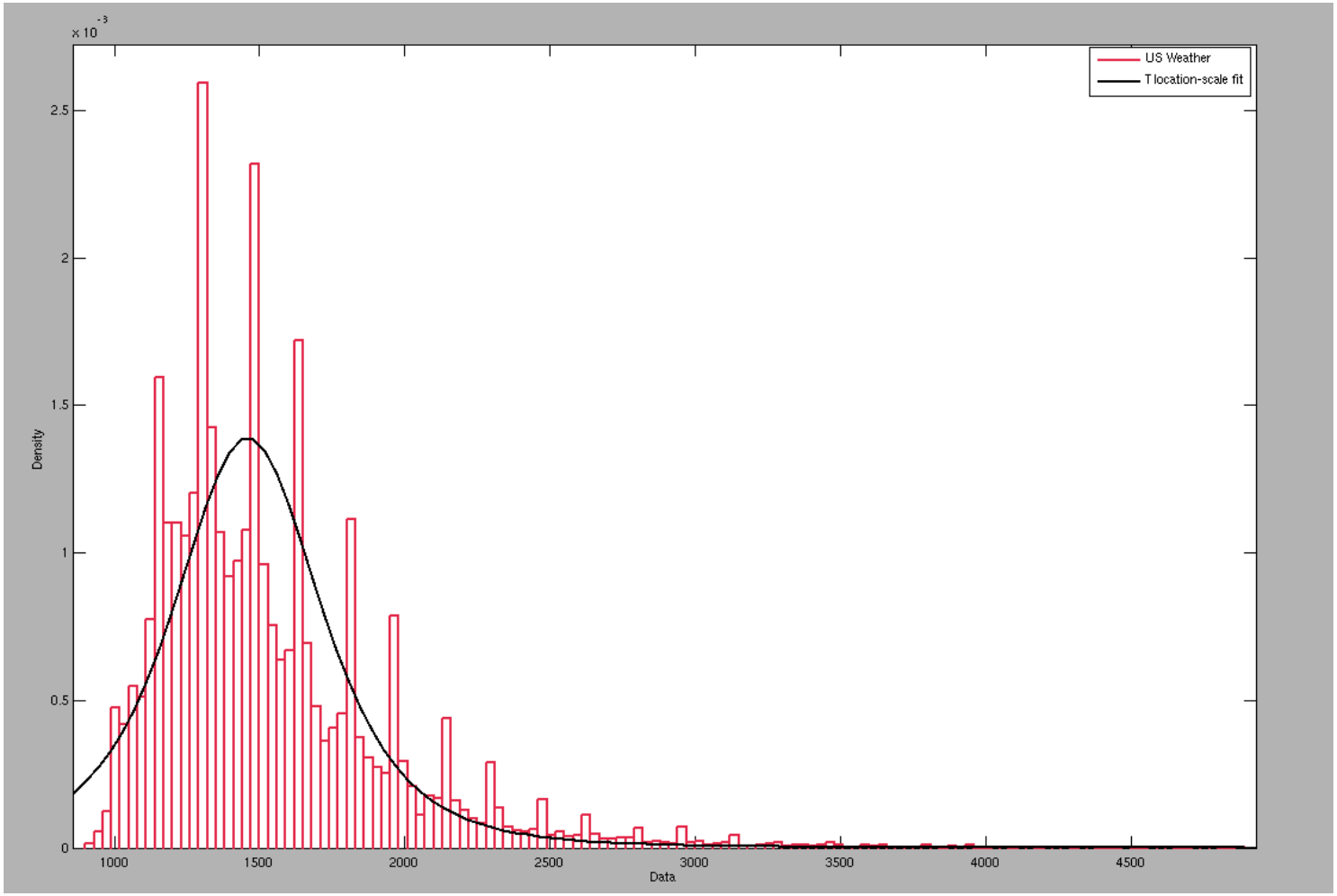
Event Structure for CarOnLine Example



Modelling Delays

- Different sites in CarOnLine equipped with delay behaviour of "real" Web Services over the internet.
- Response times (20K values) from six Web Services recorded.
  - Fast : Delay range 60-300ms
  - Slow : Delay range 2000 and 8000 ms
  - Moderate : Delay range 800-2000ms.
- Density fitting on observed delays.
- Delays for internal orchestration actions taken to be zero

T-Location Scale fit on a Weather Service delays



Simulation Modes

- Bootstrap Mode - Uniformly pick values from the set of 20,000 delays during each simulation.
- Random Sampling Mode - Sample the T-Location Scale distributions with parameters from best fit curves.

Comparing Probabilistic Contract Composition with Classical Contract Composition: $\delta \ll D$

Classical	Probabilistic
1. Assign to each site S a quantile q_S such that $\prod_S q_S = 95\%$ 2. For each site S, select delay D_S associated to quantile q_S 3. Composing the D_S using max+ rules yields D. D is promised by the orchestration for at least 95% of the cases	1. Assign to each site S its probabilistic contract 2. Compose them 3. Look at the overall distribution for the orchestration and chose δ corresponding to 95% quantile

Results (for a 94.53% quantile): safe overbooking

- Simulations were run on the CarOnLine example.
- 100,000 simulations performed in each simulation mode.

Mode	Probabilistic	Classical
BootStrap	23,189	44,243
T Location Dist	14,658	1,469,539

- All values are in milli seconds.
- The time taken for the 100,000 simulations in the bootstrap mode was 37.74 sec and in the T Location-sampling mode was 42.13 sec.

Conclusions and Perspectives..

- Results show that well sound overbooking is possible during contract composition
- Our approach establishes the foundations for contract monitoring based on statistically sound techniques

- Handling more QoS parameters (availability, throughput)
- Estimating the contribution of transport to QoS
- Reliable models: need more “real” performance data