Parallel Session 6:

Programming Models

Ugo Montanari
Dipartimento di Informatica
Università di Pisa

in collaboration with

SENSORIA Teams, WP2 and WP5

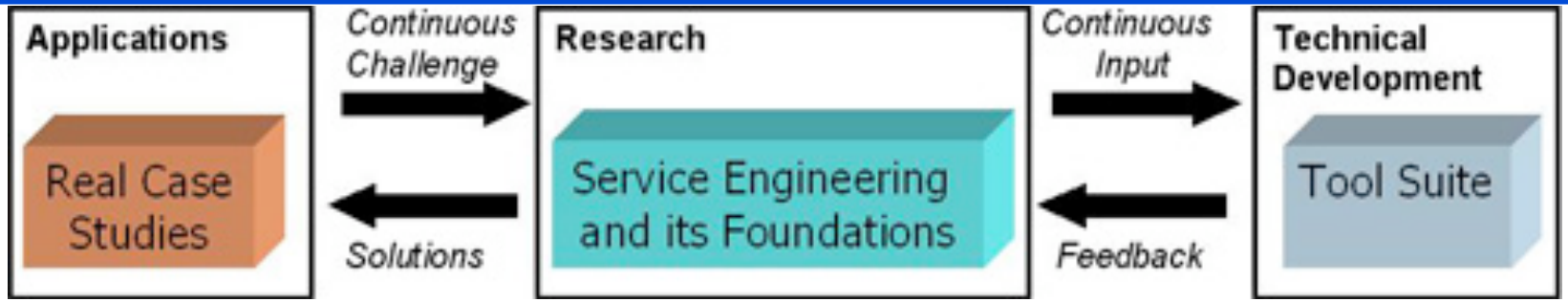
Outline

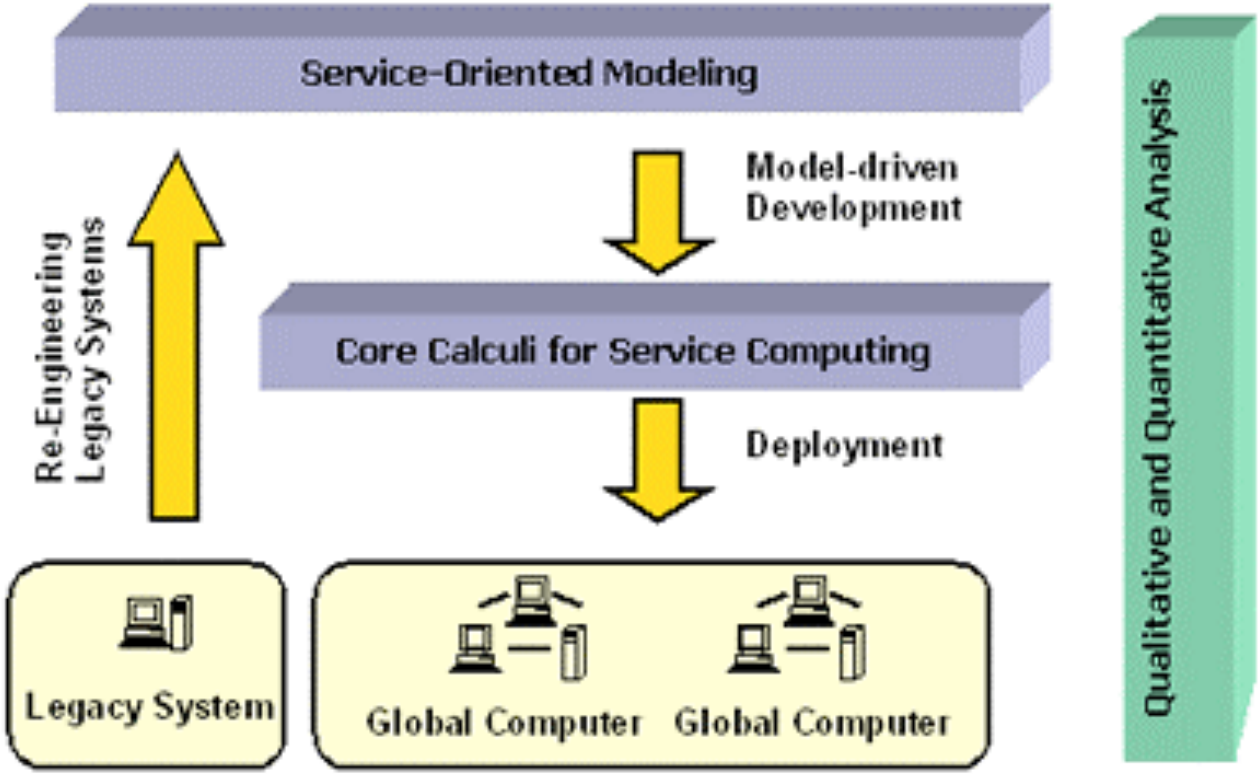
- Web Services
- Some Requirements for Programming Models
- Session Based Calculi
- Committed Negotiations
- Constraint Programming for Quality of Service
- Software Architectures as Types

Web Service Scenario

- Largely distributed systems
  - Wide Area Network (WAN)
  - Global Computing (GC)
  - Web programming and service-oriented computing
- Asynchronous communication
- Transactions, contracts, negotiations, decisions, agreements, choices
  - Causality, concurrency and distribution
- Language primitives and formal models
  - Coordination, orchestration, choreography
- Quality of service and service level agreements

SENSORIA http://sensoria.fast.de/





Models of Computation

- Useful for:
  - PL design and formal semantics
  - Formal methods & verification
  - Software architectures
- Must handle:
  - Distribution, concurrency, network awareness
  - Mobility
  - Open endness, service publication, discovery and binding
  - Negotiation
  - Security
- Existing models often inadequate

Session- Based Calculi

- Explicit notions of service definition, service invocation and session handling
- Structured orchestration of services
- Multiple session instances
- Termination handlers for closing (multiple) sessions
- ORC by J. Misra et al., e.g. CONCUR 2006
- SCC a Service-Centered Calculus, SENSORIA Team, WS-FM 06

Committed Negotiations

- Local and global resources
- Local sub-contracts and decisions
- Global results posted upon commit
- Abort of ongoing contracts
  - All participants must be informed
  - Compensations can be activated
- Either abort or commit (no divergence)?
- Dynamic joining of participants
  - Contracts can be merged
- Nested structure of contracts

Our Proposal

- committed JOIN
  - PDL presentation
  - Non ACID
  - Multiway
  - Open Nesting
  - Flexible
  - Split / Join
  - Programmable commit / abort / compensation
  - Concurrency and distribution
  - Distributed 2PC
  - Different levels of abstraction

Service Invocation as Constraint Solving

- Web services published, invoked with suitable constraints
  - more than two sites involved e.g. when routing or access paths are involved
  - a site cannot anticipate the requests of the other sites or solve the global constraint that arises in this way
  - suitable network services must be provided to do this job
- Non-crisp situations: a constraint
  - solved in a yes or no way
  - solved with some cost, or some probability, or according to several criteria at the same time => constraint semirings
- Networks of constraints
  - Graphical presentations of the constraints, distributed solution algorithm

Constraint Programming with C-Semirings

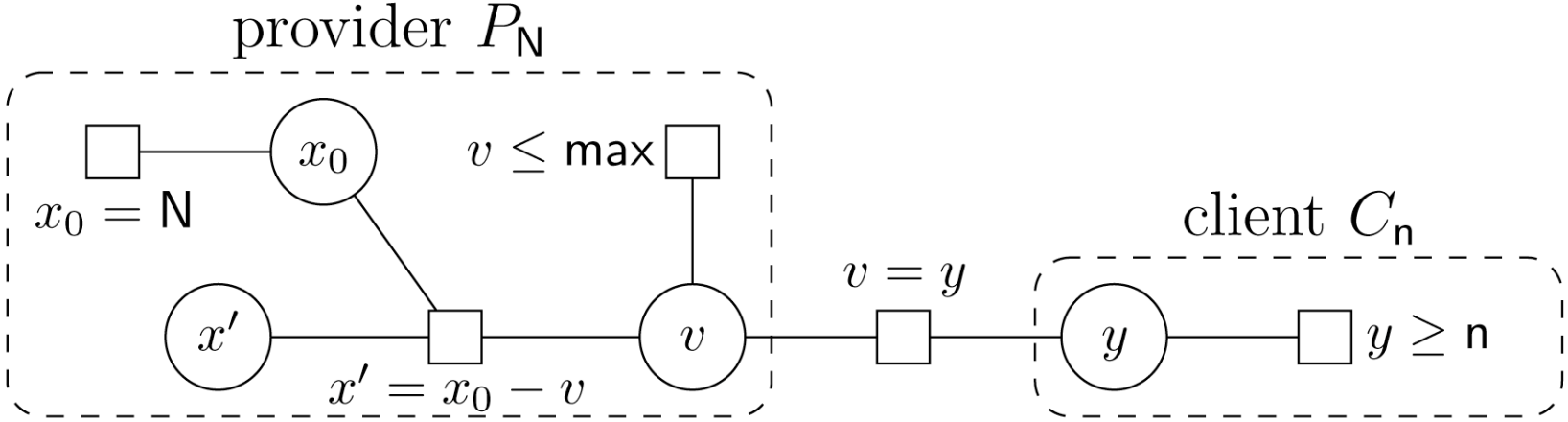
- Logical, fuzzy, optimization (tropical), probability, hierarchy, capability semiring
- Several existing algorithms can be extended to c-semirings
- Constraint Logic Programming (CLP(X)) can be extended to c-semirings
- Cartesian product of c-semirings is a c-semiring (it is not the case for booleans)
- Functional domains, power domains preserve the c-semiring structure and allow for the same logic, algorithms
- Concurrent constraint programming for network aware programming

Cc-pi, A Language for Contracts

Prefixes	$\pi ::= \tau \mid \bar{x}\langle\tilde{y}\rangle \mid x\langle\tilde{y}\rangle \mid \texttt{tell}\, c \mid \texttt{ask}\, c \mid \texttt{retract}\, c$
Unconstrained Processes	$U ::= \mathbf{0} \mid U \mid U \mid \sum_i \pi_i.U_i \mid (x)U \mid D(\tilde{y})$
Constrained Processes	$P ::= U \mid c \mid P \mid P \mid (x)P$

Cc-pi, A Language for Contracts

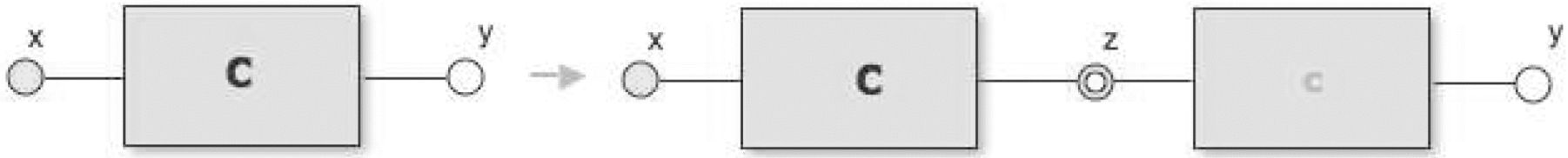
$$P_{\mathsf{N}} = (x_0)\,(\texttt{tell}\ (x_0 = \mathsf{N}).Q(x_0))$$
$$Q(x) = (v)\,(x')\,(\texttt{tell}\ (x' = x - v).\texttt{tell}\ (v \leq \mathsf{max})\,).c\langle v\rangle.Q(x')).$$
$$C_{\mathsf{n}} = (y)\,(\texttt{tell}\ (y \geq n).\bar{c}\langle y\rangle.\tau.\texttt{retract}\ (y \geq n).\texttt{tell}\ (y = 0)\,).$$



A Provider and Client example

Deriving Architectural Styles

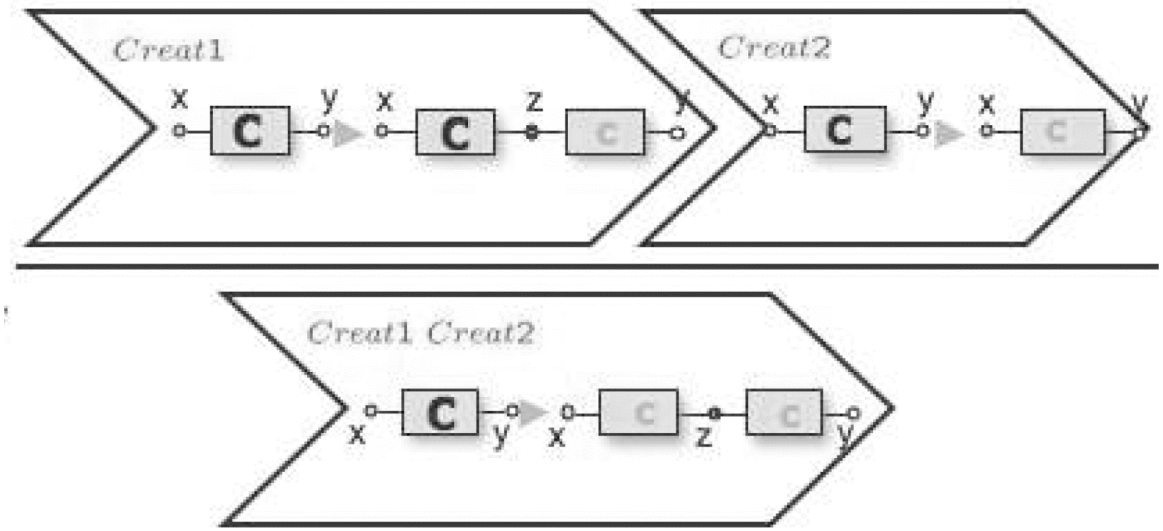
$$Creat1 \stackrel{def}{=} x, y,\ X : C \vdash \nu\ z.\ X(x, z) \mid c(z, y) : C$$



A refinement production for a pipeline architecture

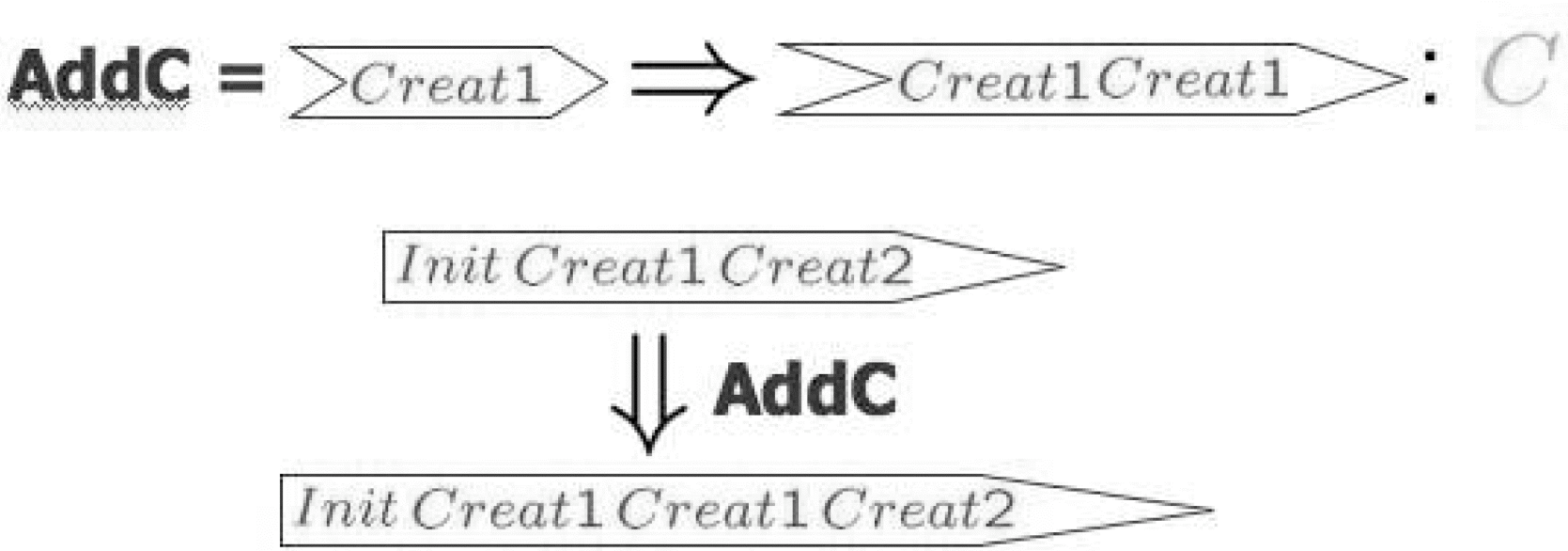
Architectural Programming

$$\frac{J = \vec{x}, \Delta_1 \vdash H : A \quad R = \Gamma, \Delta_2, X : A \vdash \nu\vec{w}.G \mid X(\vec{z}) : C}{(RJ) = \Gamma, \Delta_2, \Delta_1 \vdash \nu\vec{w}.G \mid H\left[\vec{z}/\vec{x}\right] : C} \qquad |\vec{x}| = rank(A)$$



An inference rule for composing two productions

Reconfiguration



A reconfiguation transformation between two proofs