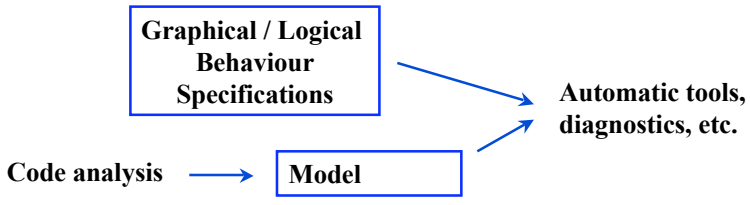
Verification of Distributed Applications

Eric Madelaine
work with
Isabelle Attali, Tomás Barros, Rabéa Boulifa,
Christophe Massol, Alejandro Vera

OASIS Project: INRIA, CNRS-I3S, UNSA
Nov. 2004

Goal

Automatic verification of properties of distributed systems.



Behaviour properties :

communication events (with values)
deadlocks, reachability, temporal ordering of requests.

Challenges

- Specification language :
  - » usable by non-specialists
- Automatic verification :
  - » construction of models from source code
  - » integrated software
- Standard, state-of-the-art model checkers :
  - » finite state models,
  - » hierarchical models, compositional construction

Challenges (2)

Complexity of (Distributed) Software Verification

- Classical approaches:
  - BDDs, partial orders, data-independance, symmetry, state abstractions
  - Value-passing systems, bounded model-checking, data abstractions
  - Hierarchical construction, compositional reduction techniques

Safe Abstractions
Parameterized Models
Automatic Tools

- Parameterized / Infinite systems
  - ad-hoc, problem-specific solutions (induction, widening, etc.)

Plan

- Parameterized hierarchical models
- Graphical Specification Language
- Extracting Models from ProActive Code
- Compositional verification
- Components

Model (1) : Synchronisation Networks

- Labelled Transition Systems (**LTS**) : <S,s0,L, → >
- Synchronisation Network (**Net**) :
  - operator over transition systems (finite arity, arguments with sorts)
  - synchronisation vectors : Ag <- [*, *, a3, *, a4, a5, *]
  - dynamic synchronisation : transducers
- Synchronisation product :
  builds a global LTS from a Net of arity *n*, and *n* argument LTSs.

- Arnold 1992 : ***synchronisation networks***
- Lakas 1996 : ***Lotos open expressions***
- Boulifa, Madelaine 2003,
  Model generation for distributed Java programs, **Fidji'03**

(2) Parameterized Transition Systems

- Process Parameters :
  - denotes families of LTSs.
- Variables :
  - associated to each state, assigned by transitions.
- Simple (countable) Types :
  - booleans, finite enumerations, integers and intervals, records.
- Parameterized LTS (**pLTS**) $<K,S,v_s,s_0,L, \rightarrow >$

  with parameterized transitions : $\xrightarrow{[b]\ \alpha(x),\ x':=e(x)}$

(3) Parameterized Networks

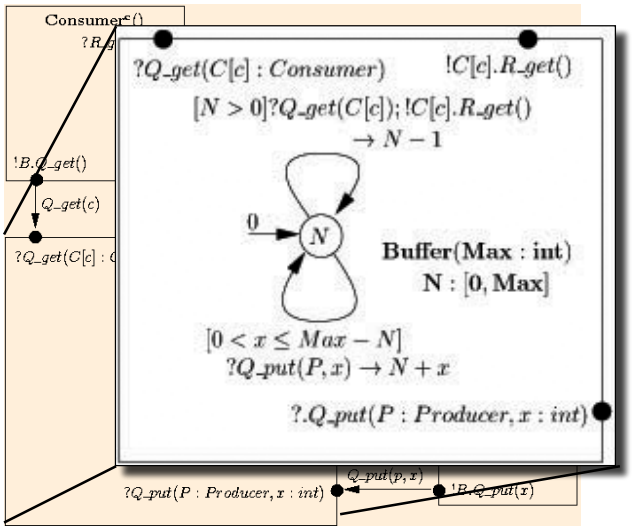
- Synchronisation Network (**pNet**)

  $<pA_G,\ H=\{pI_i,K_i\},\ pT = <K_G,T_T,t_0,L_T, \rightarrow >>$
  - global action alphabet pAg,
  - finite set of arguments, each with sort pI_i and params K_i, corresponding to as many actual arguments as necessary in a given instantiation,
  - parameterized synchronisation vectors

    pAg <- [*, *, a3(k3), *, a4(k4), *]
- Instantiation : for a finite abstraction of the parameters domains D_v

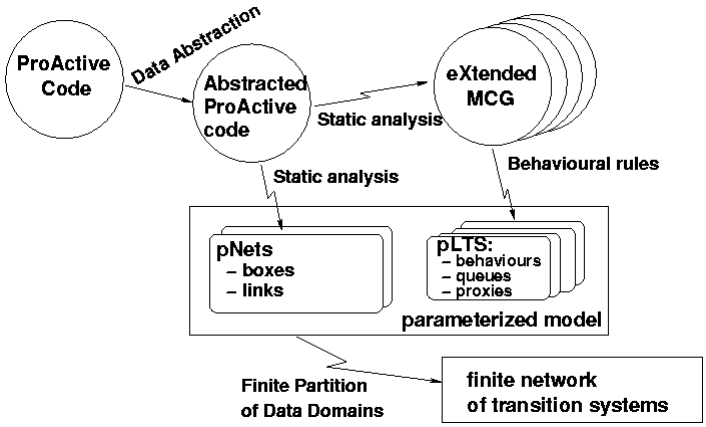
  $pLTS \times D_v \rightarrow LTS$
  $pNet \times D_v \rightarrow Net$ } Finite Network

Graphical Models



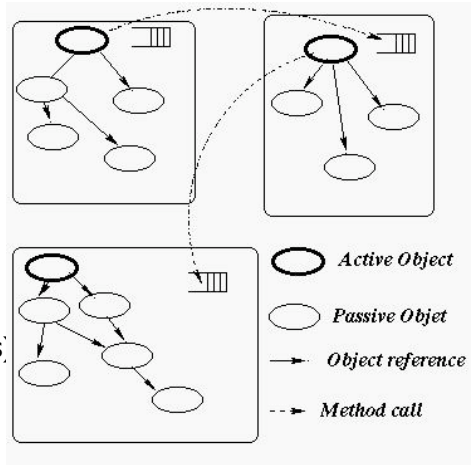
- Graphical Models :
  - Less powerful than general parameterized models (static, finite, 1-1 communication).
  - More intuitive : can be used by non-specialists.
- Models Generated from Code :
  - Very expressive : encode various schemes of communication.
- Use common model to compare specification versus implementation (preorders).

Extracting models : principles



ProActive : distributed activities

- Active objects communicate by Remote Method Invocation.
- Each active object:
  - has a request queue (always accepting incoming requests)
  - has a body specifying its behaviour (local state and computation, service of requests, submission of requests)
  - manages the « wait by necessity » of responses (futures)



ProActive : High level semantics

- Independence wrt. distribution
- Guarantee and Synchrony of delivery :
  - RdV mechanism ensures the delivery of requests, and of responses.
- Determinism / Confluence :
  - Asynchronous communication and processing do not change the final result of computation.

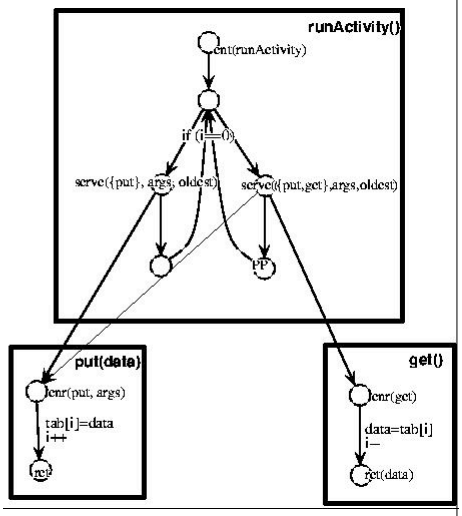
ASP Calculus: **D. Caromel, L. Henrio, B. Serpette, "*Asynchronous and Deterministic Objects*", POPL'2004**

Step 1: Front end abstractions

- Various methods for simplifying source code, with respect to a (set of) properties to be proven:
  - **Data abstraction** : transform the application data domains into "simple types".
  - **Slicing** : only keep variables and instructions influencing the property of interest.
- The BANDERA toolset offers modules for slicing and data abstraction. We have adapted them to deal with ProActive code.
- We use JIMPLE as an intermediate code for defining our static analysis functions (simpler than bytecode).

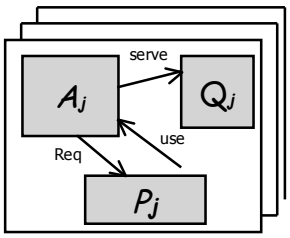
Step 2 : Parameterized Call Graphs

- control flow : class analysis + method calls
- data flow : sequences of instructions (bytecode level)
- distribution : identification of active objects in the code: activities, remote calls, futures.
- Complex static analysis :
  - class analysis
  - alias analysis
  - approximation of object topology
  - simulation of generated code.



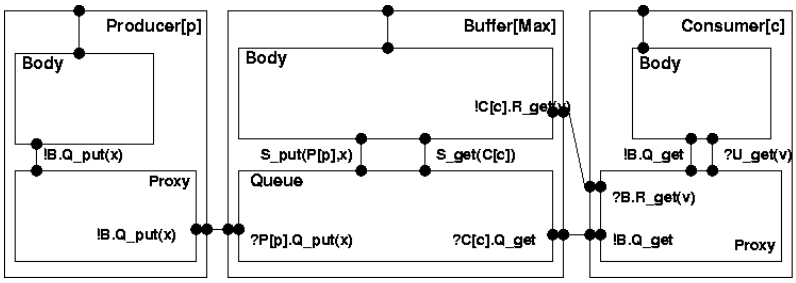
Step 3a : Model generation
Global Network

- Static topology : finite number of parameterized activities.
  - Identify parameters
  - Build boxes and links for each activity



- For each Active Object Class :
  - Body : parameterized network of LTSs
  - local method calls : synchronisation messages
  - remote calls : "wait by necessity" using proxy processes
  - requests queue : the main potential blow-up...!

Step 3b : Model generation
Global Network

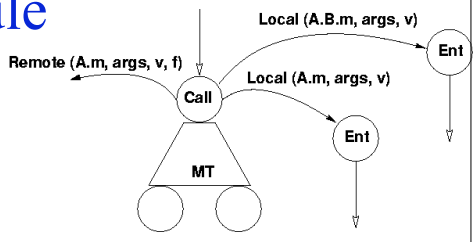


• Property : for each distributed active object class, starting from source code with abstracted data (simple types), our procedure terminates and builds a finite parameterized model.

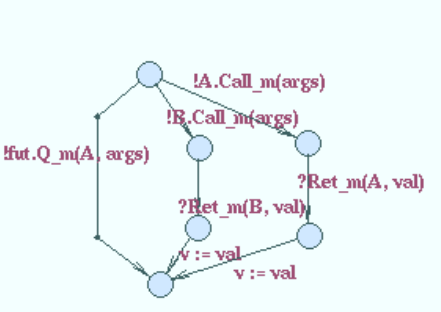
Step 3c : Model generation
Method LTS

- One **pLTS** for each method in the Active Object
- For each method :
  - a residual algorithm for crossing the pMCG
  - generates a parameterized LTS of linear size (each pMCG node is crossed only once)
  - imprecision of the static analysis results in non-determinism.

Example : Call rule



- Approximations from static analysis:
  - class imprecision
  - remote/local objects

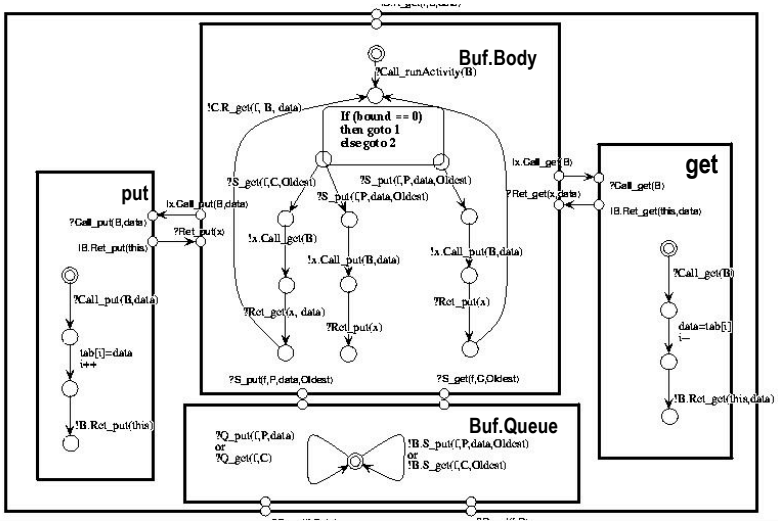


• Local calls : sends an activation message to the corresponding process,

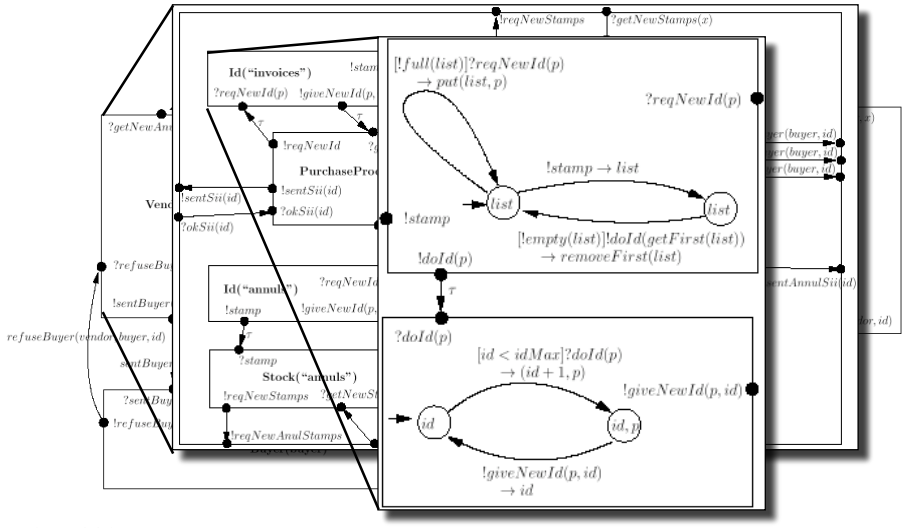
• and waits for the return message.

• Remote calls : sends a request message to the proxy and to the remote object.

Buffer Network



Large case-study: Electronic Invoices in Chile



Electronic Invoices in Chile

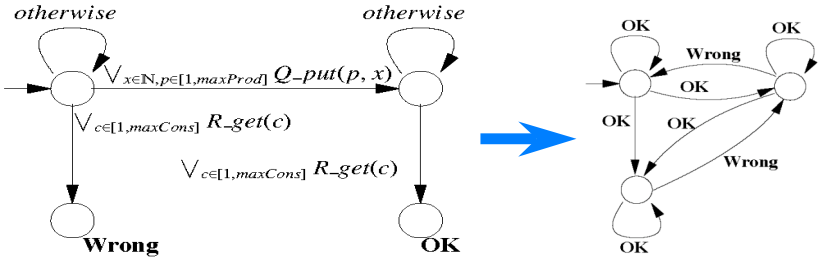
Barros, Madelaine "***Formalisation and Verification of the Chilean electronic invoice system***", SCCC'04 Arica and INRIA report RR-5217, june 2004.

- 15 parameterized automata / 4 levels of hierarchy
- *state explosion*: grouping, hiding, reduction by bisimulation

N^r	*idMaxInv*	*idMaxCancel*	*maxStockInv*	*maxStockCancel*	*pchPros*	*buyerSet*	*vendorSet*	Vendor	Buyer	SII	Global
1	1	1	1	1	1	1	1	140/860	6/9	64/348	752/2,816
2	3	1	1	1	1	1	1	5,404/37,162	216/972	16,384/168,960	58,960/290,208
3	1	3	1	1	1	1	1	140/860	6/9	64/348	752/2,816
4	1	1	5	1	1	1	1	420/3,456	6/9	64/476	2,256/10,896
5	1	1	1	5	1	1	1	420/3,432	6/9	64/476	2,256/10,752
6	1	1	1	1	5	1	1	51,428/347,944	6/9	64/348	278,256/1,199,648
7	1	1	1	1	1	3	1	404/2,500	6/9	64/348	3,088/11,824
8	1	1	1	1	1	1	2	140/860	36/108	4,096/44,544	565,504/4,235,264
9	3	3	1	1	1	1	1	8,812/63,710	216/972	16,384/168,960	90,064/462,208
10	1	1	1	1	1	1	3	140/860	216/972	262,144/4,276,224	*unknown*
11	2	2	2	2	2	2	2	4,950/38,697	1,296/7,776	*unknown*	*unknown*

Parameterized Properties

- Logical parameterized LTS

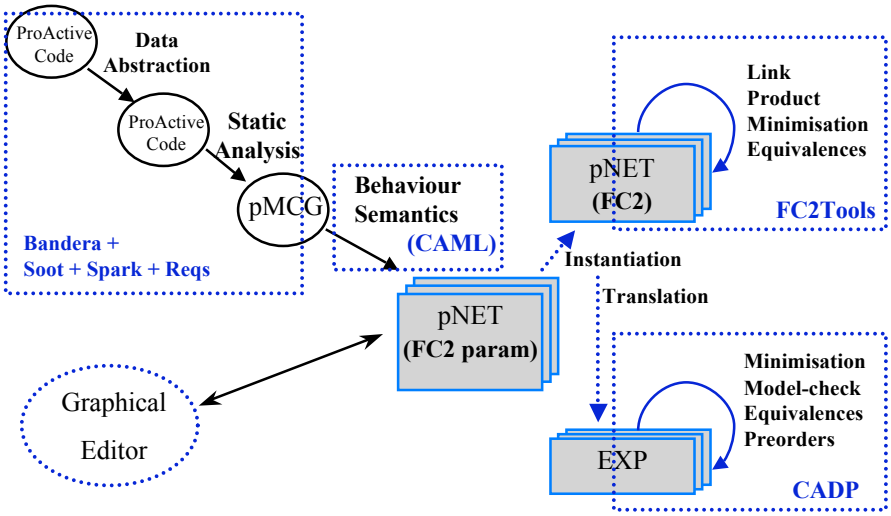


- Parameterized temporal logics

$$\forall c_{c\in[1,maxCons]} / AG(Q\_get(c) \Rightarrow AF\ R\_get(c))$$

True/False + diagnostic

Prototype tools

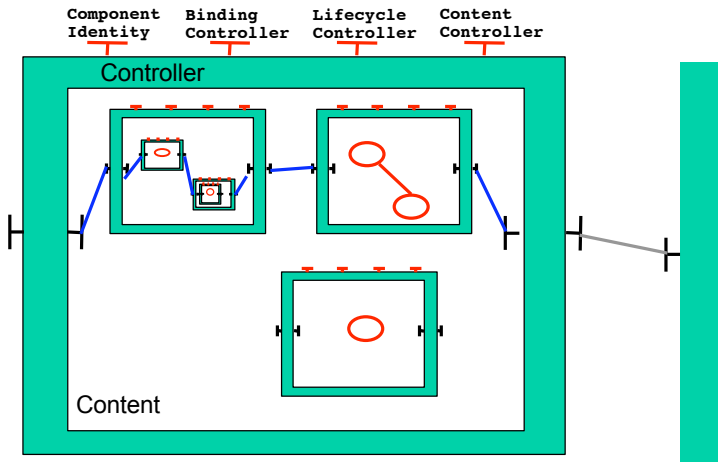


Distributed Componants

- Context :
  - Very few established models for hierarchical distributed components
  - Provide/Require interfaces
    - » Correctness = standard typing of RPC
  - Research ongoing on Behavioural Typing
- Tracks :
  - Fractal Hierarchical model
  - Fractal / ProActive Distributed Components
  - pNets as behavioural types for composition.

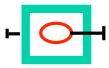
Fractal hierarchical model :

composites encapsulate primitives, which encapsulates Java code

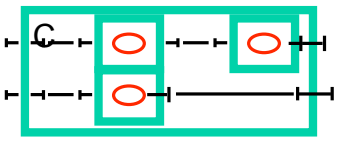


Fractal + ProActive Components for the GRID

An activity, a process, …
potentially in its own JVM

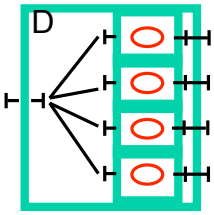


1. Primitive component
Java + Legacy



2. Composite component

Composite: Hierarchical, and Distributed over machines

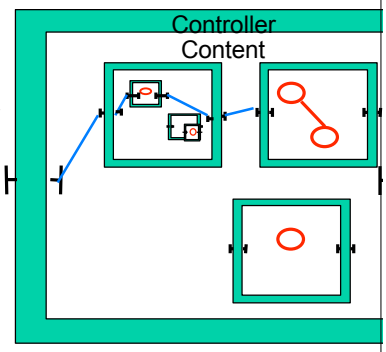


3. Parallel and composite component

Parallel: Composite + Broadcast (group)

Components : correct composition

- Behaviour is an essential part of a component specification.
- Model of components :
  - primitive = pLTS
  - composite = pNet
  - state-less component = static pNet
  - controller = transducer
- Correctness of composition :
  - implementation preorder ?



Conclusions

- Parameterized, hierarchical model.
- Graphical specification language.
- Validated with a realistic case-study.
- Generation of models from ProActive code.
- Ongoing development of prototype tools, incorporation within a verification platform.

(**ACI-SI Fiacre : INRIA-Oasis, INRIA-Vasy, ENST-Paris, SVF**)

Perspectives

- Refine the graphical language, extend to other ProActive features, extend the formalization of abstractions.
- (Direct) parameterized verification.
- Behavioural specifications of components, correct compositions.

http://www-sop.inria.fr/oasis/Vercors

Thank you

http://www-sop.inria.fr/oasis/Vercors