

# **Service Contracts in a Secure Middleware for Embedded P2P Systems**

**Antonio Brogi**

Department of Computer Science  
University of Pisa, Italy

*Joint work with F. Benigni, S. Corfini, T. Fuentes*

FLACOS'08, Malta, 27-28 November, 2008

# Introduction

- ❑ **Mobile** peer-to-peer systems
  - ❑ All network elements can act both as service consumers and as service providers, enormous potential for distributed applications ...
    - communication often relies on dynamic ad-hoc networks
    - constantly changing topology
    - frequent connections and (possibly unexpected) disconnections
  - ❑ Two important derived issues:
    - “Exception is the rule”
    - Security
  - ❑ Need of appropriate middleware to abstract from low-level issues while developing P2P applications

# The SMEPP project



- ❑ Ongoing EC project “**S**ecure **M**iddleware for **E**mbedded **P**eer to **P**eer Systems” ([www.smepp.org](http://www.smepp.org))
- ❑ Goal: To develop a middleware for EP2P that will have to be
  - ❑ **secure**
    - supporting and facilitating the use of different levels of security for different (parts of) applications
  - ❑ **versatile**
    - adaptable to different devices (from laptops to PDAs and smart phones to sensor networks)
    - usable in different application domains (from critical systems to domotics to consumer entertainment)
  - ❑ **service-oriented**
    - to enable programming applications at a suitable abstraction level

# The SMEPP project

- The analysis of the state-of-the-art highlighted that available P2P models:
  - do not provide a simple, high-level service model to ease the development of P2P applications, or
  - do not model all concepts emerged in the definition of SMEPP's middleware and application requirements, or
  - do not provide a (formal) abstract language that can be used for simulating and verifying the behaviour of peers and services, as well as for application prototyping
- SMEPP therefore introduced
  - 1) A service-oriented **model** and a set of **primitives** for programming secure (E)P2P applications
  - 2) A simple modelling language (**SMoL**) to specify peers and services by orchestrating SMEPP primitives

# SMEPP primitives

Key notions of the model

- **group** of peers
- **service** offered by peers or groups
- **security-awareness**

```
// Peer Management
pId newPeer(creds)
pId getPeerId(id?)
pId[] getPeers(gId)

// Group Management
gId createGroup(grDescr)
gId[] getGroups(grDescr?)
grDescr getGroupDescr(gId)
void joinGroup(gId, creds)
void leaveGroup(gId)
gId[] getIncludingGroups()
gId getPublishingGroup(id?)

// Service Management
<gSid, pSid> publish(gId, contract)
void unublish(pSid)
<gId, gSid, pSid>[]
    getServices(gId?,pId?, contract?, creds)
contract getServiceContract(id)
sessId startSession(sId)

// Message Handling
out? invoke(eId, opName, in?)
<cId, in?> receiveMessage(gId?, opName)
void reply(cId, opName, out?, fName?)

// Event Handling
void subscribe(evName?, gId?)
void unsubscribe(evName?, gId?)
void event(gId?, evName, in?)
<cId, in?> receiveEvent(gId?, evName)
```

# SMoL (SMEPP Modelling Language)

## □ Objective

- Provide a high-level language for specifying how to orchestrate SMEPP primitives into peer or service code
  - to simplify the time-consuming and error-prone task of specifying the interactions of a complex P2P system.
- Define formal semantics for such a language
  - to enable the simulation and the analysis of the behaviour of peers and services
  - in order to feature the possibility of developing not only secure, but also *a priori* verified SMEPP specifications
- Simplify the generation of executable code by means of automatic translators
  - eg, the prototype SMoL2Java compiler

# SMoL (SMEPP Modelling Language)

- Inspired by WS-BPEL 2.0  
(the OASIS standard to orchestrate several WSDL services)  
but removing from WS-BPEL the constructs (not needed for SMEPP) that make its semantics hard to be analysed, like **synchronisation links** or **compensations**

# SMoL syntax

```
basicCommand ::= P | empty() | wait(for?,until?,repeatEvery?) |
              throw(faultName,faultVariable?) |
              catch(faultName) | catchAll() | exit()

C ::=
  basicCommand |
  Assign from to |
  Sequence(C,...,C) |
  Flow(C,...,C) |
  While cond Do C | Repeat C Until cond |
  If cond Then C Else C |
  Pick(guard → C + ... + guard → C) |
  InformationHandler (/*while*/ C Do guard → C + ... + guard → C) |
  FaultHandler (/*while*/ C Do fhGuard → C + ... + fhGuard →C)

guard ::= receiveMessage(...) | receiveResponse(...) |
         receiveEvent(...) | wait(...)
guard ::= receiveMessage(..) | receiveEvent(..) | wait(..)
fhGuard ::= catch(..) | catchAll(..)
```



# Summary of contributions so far

## □ Contributions

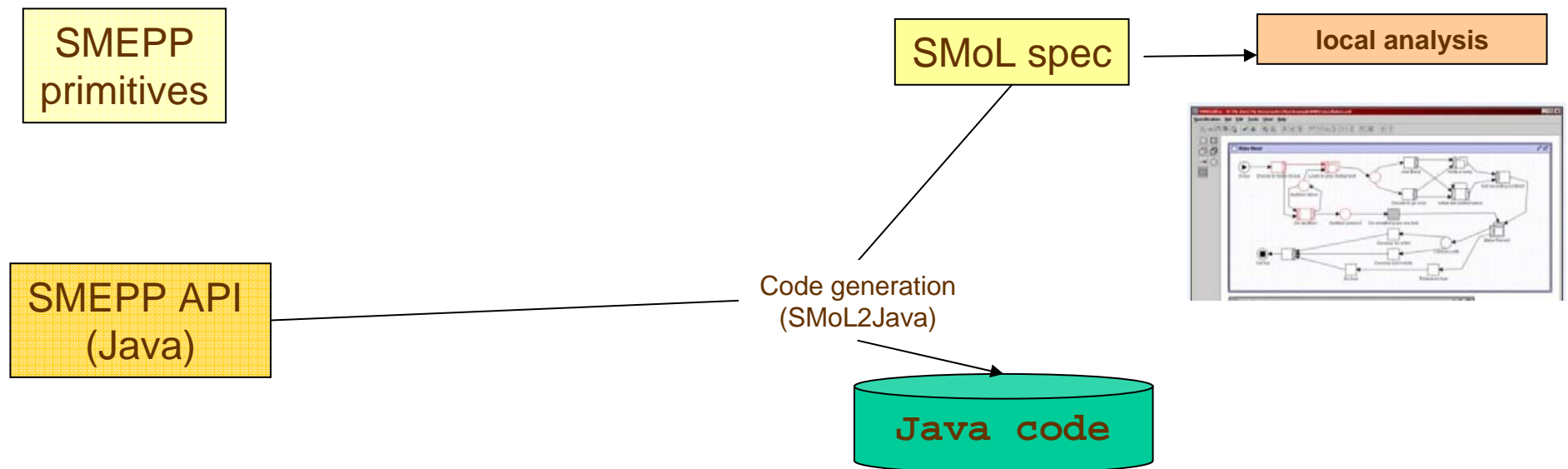
1) A service-oriented **model** and a set of **primitives** for programming secure (E)P2P applications

2) A simple modelling language (**SMoL**) to specify peers and services by orchestrating SMEPP primitives

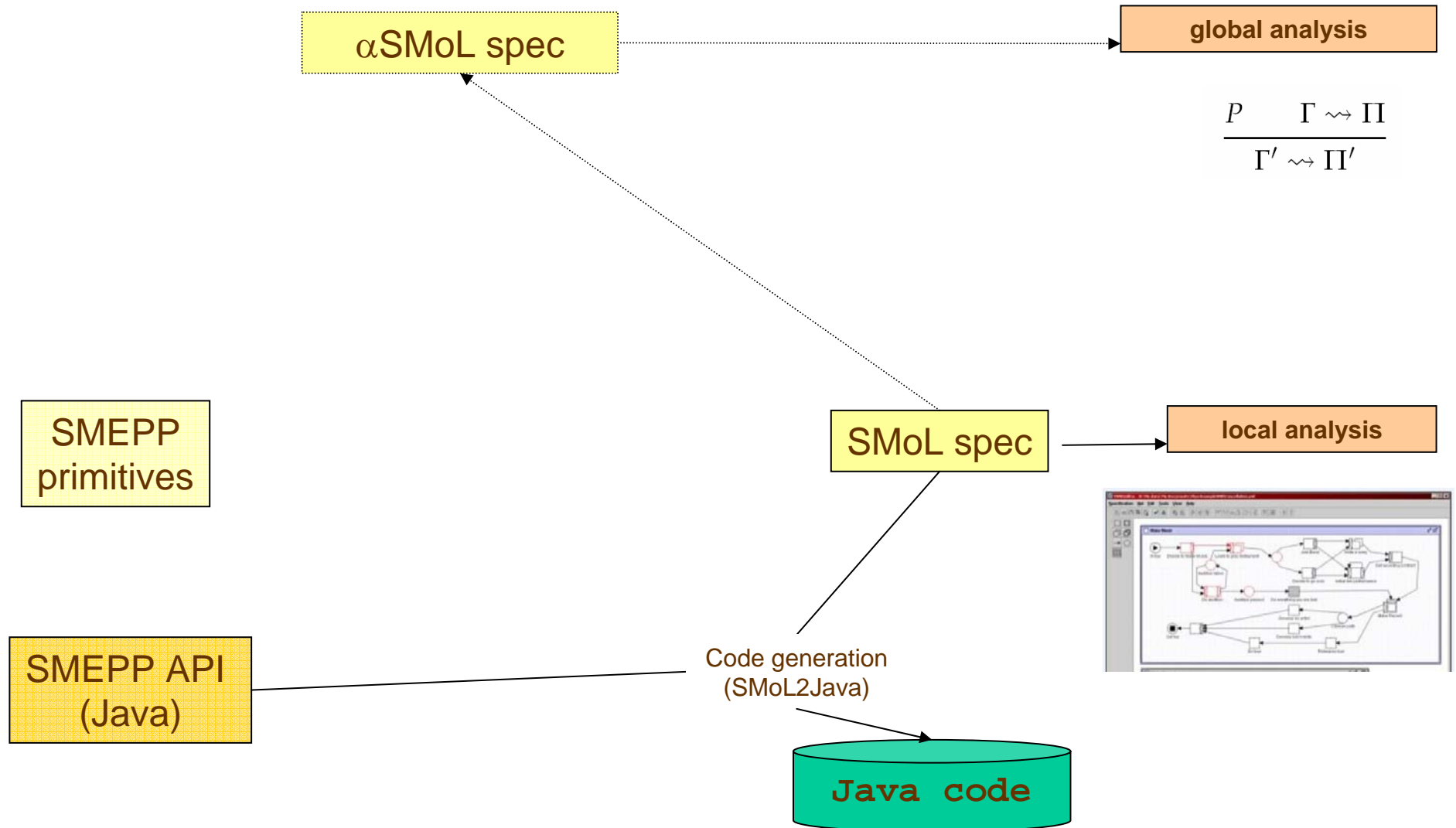
3) Semantics for SMoL

- a **transformational semantics** for SMoL – via a pattern-based compositional translation of SMoL programs into YAWL workflows
- a **sequent-calculus semantics** for an abstract version of SMoL - it allows to determine whether a set of peer and service specifications can be executed together without locks

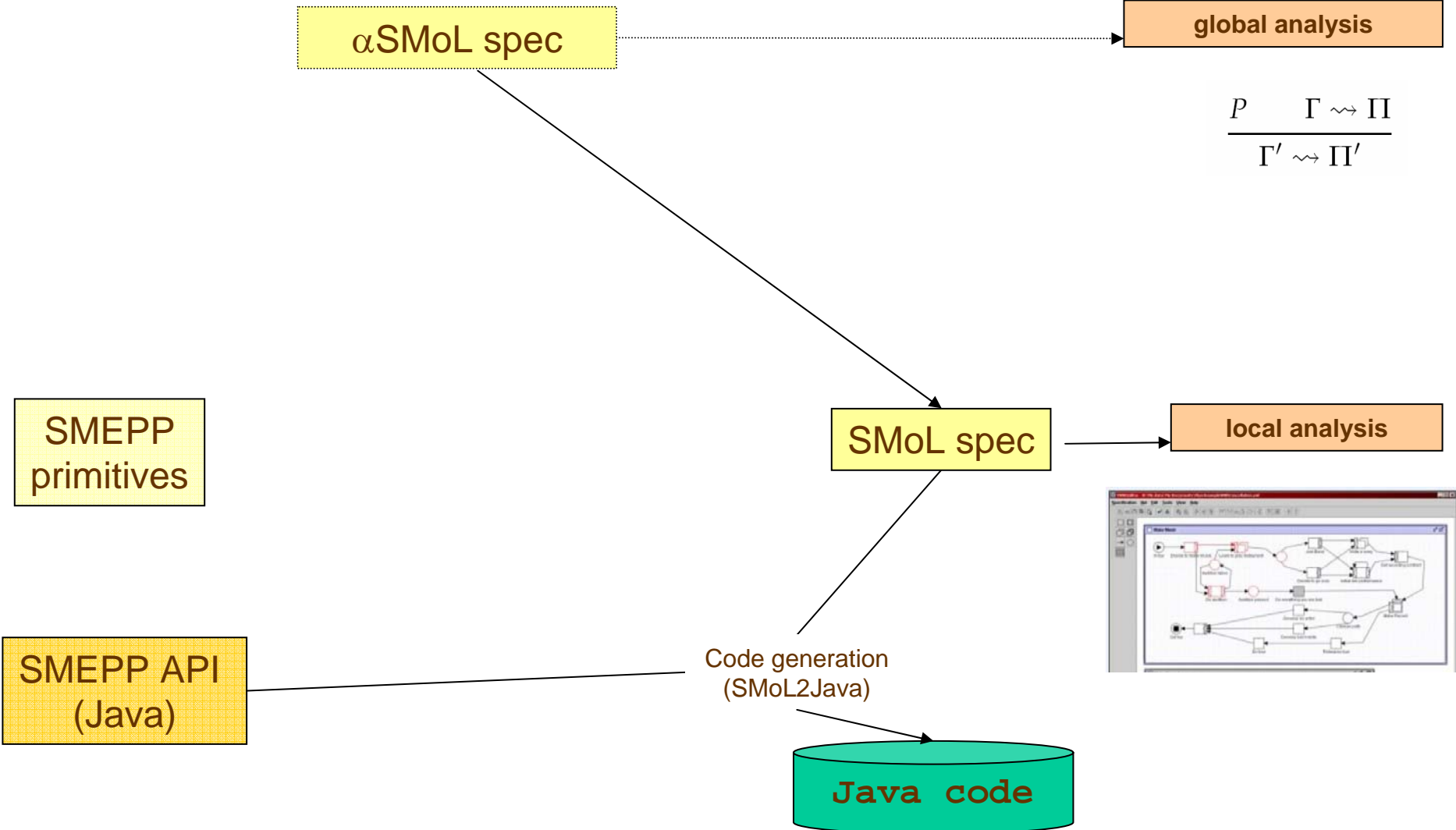
# Incremental software development in SMEPP



# Incremental software development in SMEPP



# Incremental software development in SMEPP



# SMEPP service contracts

- SMEPP services are associated with:
  - **service contracts** - which provide standard descriptions of SMEPP services - and with
  - **service groundings** - which provide metadata needed to correctly interoperate with (third-party) services
    - communication protocols, message formats, port numbers, etc.
    - grounding information accessible only by the provider's MW
- Service contracts are the key ingredients of the SMEPP mechanisms for service publication, discovery, and analysis

$[\text{contract}]_{\text{SMEPP}}$  = self-presentation of a service

FLACOS'08 discussions → Need of clearly distinguishing *self-presentations of services* from *inter-party agreements contracts*

# SMEPP service contracts

## □ **Signature information**

- Operations, events, parameters, types
- Ontological annotations [optional]
- Signature information used for discovery, needed for service invocation
- Expressed with (a subset of) WSDL2.0 [and OWL]

## □ **Behaviour information**

- Term representing the interaction behaviour of the service (orchestration of SMEPP primitives)
- Behaviour information is optional (stateless services)
- Behaviour information used for discovery and analysis
- Expressed as a SMoL specification

## □ **Properties [optional]**

- Service categorization according different criteria (e.g., geographical, business type)
- Used for discovery
- List of properties of the form *<category,name,value>*  
*category* can be a reference to a taxonomy

## □ **QoS information [optional]**

- List of QoS parameters, each can include
  - *Name, Value*
  - *Domain* - classification of the information (e.g., Runtime-related, Transaction-Support, Security-Level, Cost-related, etc.).
  - *QoSImpact* - describes the way in which a variation or unfulfillment of the QoSParameter would affect the performance and QoS of the service
  - *QoSMetric* – measure unit with which the QoSParameter can be measure
  - *Relationship* - describes how the QoSParameter can affect other QoSParameters
  - *Aggregations* - compositional rules applied to the QoSParameter, used to describe compound QoSparameter....
- Ontological annotations [optional]
- Format borrowed from AMIGO project
- Used for discovery (and for monitoring)

# Simple example of SMEPP contract

```
Begin_Contract:
Profile:
  ServiceName: TemperatureReader
  ServiceCategory: EnvironmentMonitoring

Signature:
  Request-response operation temperature getTemp()

Behaviour:
  ServiceType: state-less
  Process:
    Sequence
      <callerId> = receiveMessage("getTemp")
      t = opaque // measure ambient temperature
      reply(callerId, "getTemp", t)
    End Sequence

Properties:
  [Geography::Location] = "Italy";
  [Business::Functionality] = "Environmental Sensor"

QoS:
- [Transaction_Support::integrity] = 100`%'
- [Runtime_Related::latency] = 5`sec'
  latency produces an inverselyProportional impact over performance
- [Runtime_Related::throughput] = 1000`request per hour'
  throughput produces a proportional impact over performance
- [Runtime_Related::performance] = 100`%'
  performance is a compound QoSParameter composed by latency and throughput

End Contract
```

(friendly  
syntax)

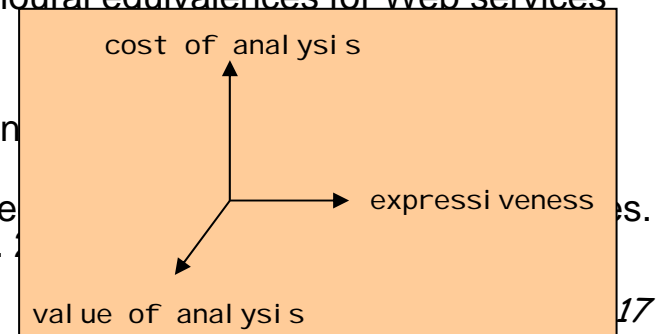
# Concluding remarks

- ❑ Service discovery = service localization + matching
- ❑ SMEPP service localization
  - ❑ group-aware, scalable and capacity-aware
  - ❑ prototype implementation based on a Chord-like overlay network
- ❑ SMEPP service discovery
  - ❑ relies on service contracts and on queries expressing partial contract specifications
  - ❑ current prototype implementation supports syntactic queries – taking into account ontological annotations, if any
  - ❑ future activities
    - provide (user-friendly) editor for contracts
    - experiment resource requirements for the different types of devices participating in SMEPP applications
    - develop semantics- and behaviour-based matching algorithms
      - identify a suitable, less expressive language (e.g,  $\alpha$ SMoL, behavioural types or even FSMs) to represent service behaviour, in order to make their inclusion in the matching feasible in the context of SMEPP
    - devise different types of matching for the different middleware configurations



# What I should have talked about, maybe

- **Contract-based service discovery** (using PNs and WFs)
  - A. Brogi, S. Corfini. Behaviour-aware discovery of Web service compositions. *International Journal of Web Services Research*, 4(3), pages 1-25, 2007.
  - A. Brogi, S. Corfini, R. Popescu. Semantics-based Composition-oriented Discovery of Web Services. *ACM Transactions on Internet Technology*, 8(4), 2008.
  - A. Brogi, S. Corfini. Ontology- and Behaviour-aware Discovery of Service Compositions. *International Journal of Cooperative Information Systems*, 17(3):319-347, 2008.
- **Contract-based service composition** (using PNs and WFs)
  - A. Brogi, R. Popescu, M. Tanca. Design and Implementation of SATOR: a Web Service Aggregator. *ACM Transactions on Software Engineering and Methodologies*. 2009. (To appear.)
  - A. Brogi, R. Popescu. From BPEL Processes to YAWL Workflows. 3rd International Workshop on Web Services and Formal Methods (WS-FM'06), volume 4184 of LNCS, pages 107–122, 2006.
- **Contract-based service adaptation** (using PAs and WFs)
  - A. Brogi, C. Canal, E. Pimentel. Component adaptation through flexible subservicing. *Science of Computer Programming*, 63(1):39-56, 2006.
  - A. Brogi, C. Canal, E. Pimentel. On the semantics of software adaptation. *Science of Computer Programming*, 61(2): 136-151, 2006.
  - A. Brogi and R. Popescu. Service Adaptation through Trace Inspection. *Int. J. Business Process Integration and Management*, Vol. 2, No. 1,9-16 2007.
  - A. Brogi and R. Popescu. Automated Generation of BPEL Adapters. In Dan and Lamersdorf, editors, *ICSOC'06*, volume 4294 of LNCS, pages 27–39, Springer, 2006.
- **Bisimulation-based compositional equivalence for Web services** (using PNs)
  - F. Bonchi, A. Brogi, S. Corfini, F. Gadducci. On the use of behavioural equivalences for Web services' development. *Fundamenta Informaticae*, 2009. (To appear.)
- **Behavioural types for (Web) services**
  - A. Brogi, C. Canal, E. Pimentel. Behavioural types and component... *3116,2004*.
  - A. Brogi, C. Canal, E. Pimentel. Behavioural types for service inter... *Electronic Notes in Theoretical Computer Science*, 180(2):41-54.



# **Service Contracts in a Secure Middleware for Embedded P2P Systems**

**Antonio Brogi**

Department of Computer Science  
University of Pisa, Italy

*Joint work with F. Benigni, S. Corfini, T. Fuentes*