Santiago, Chile - November 8th 2004

From Metaobject Protocols to Versatile Kernels for AOP

Éric Tanter

PhD Thesis Defense

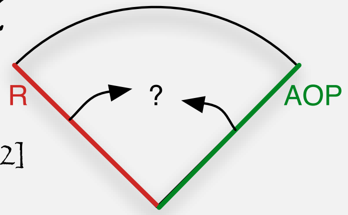
Universidad de Chile
DCC - CWR

Université de Nantes
OBASCO - EMN/INRIA

Context

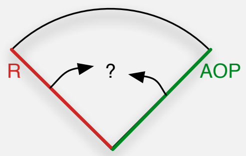
- Software design: fundamental trade-off
  - structure (modularization)
  - evolution (adaptation)
- Basic principles
  - Separation of Concerns (SoC) [Dijkstra68]
  - Information hiding [Parnas72]
- Technical means
  - modules, classes, objects: limits

Problem Statement



- Computational reflection [Smith82]
  - most general approach
  - Issues of reflection limit its acceptance
    - cost, rigidity, complexity
- Aspect-Oriented Programming [Kiczales+97]
  - More specific support for modularization
    - aspect languages
  - Sacrificing flexibility and extensibility

Thesis in a nutshell



- An operational model of reflection
  - genericity of reflection and specificity of AOP
- Versatile substrate for AOP
  - based on reflective model
  - combined with guidance of aspect languages
- Validation
  - Prototype implementation in Java: Reflex
  - Significant applications

Contents

- Thesis in a Nutshell
- Concepts: reflection & AOP
- Contributions
- Conclusions & Perspectives

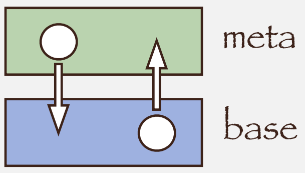
Computational Reflection

[Smith82]

- Computational system (CS) [Maes87]
  - program (text) + evaluator
- Metasystem
  - CS manipulating other programs/CS
  - evaluator, debugger, ...
- Reflective system
  - CS accessing its own metasystem

Reflection and Adaptation

- Reflection operators [FriedmanWand84]
  - reification
  - absorption



- A program can
  - observe its evaluator's state (introspection)
  - modify its evaluator (intercession)
- e.g. concurrency at the metalevel

Reflection & OOP: MOPs

- Structure metalevel interface with OO
  - get the benefits of object orientation
    - abstraction, encapsulation, localized extension
  - Metaobject Protocols (MOPs) [Kiczales+91]
- Different models [Ferber89]
  - metaclass [Cointe87], metaobject [Maes87], message reification, etc.

MOPs: Modeling Issues

- nature of metalink [Matsuoka+91]
  - individual based, group wide, hybrid...
- metalevel structure [McAffer96]
  - structural vs. operational decomposition
- granularity, locality of change [GowingCahill96]
  - fine-grained MOPs, multiple reflective models

Mastering Locality

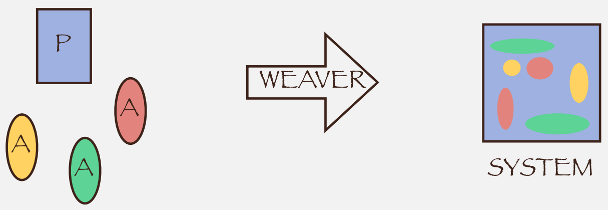
- Central tension [Kiczales92]

  "very often, the concepts that are most natural to use at the metalevel crosscut those provided at the base level"

- support crosscutting views of a system?

AOP Principles [Kiczales+97]

- Modularization of crosscutting functionalities
  - providing extra composition mechanisms
  - GP languages: procedure call
  - responsible for code tangling
- Aspects, Weaver



AOP Languages

- Join-point models [Masuhara+03]
  - join-points
    - points of reference in a base program that aspects can affect
  - means of identifying join points
  - means of effecting at join points

AOP & Reflection

"AOP is a goal, for which reflection is one powerful tool"
[Kiczales+97]

"AOP is not reflection"
[Douence04]

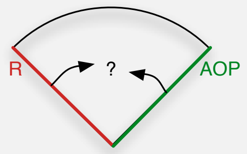
"AOP is a principled subset of reflection"
[Kiczales01]

- Relation
  - reflection can be used for aspect weaving
  - used in the early experiments on AOP
- Intrigue
  - first, ideas of reflection and MOPs
  - then, shift to "reflection-free" discourse
  - what's the reality behind? spectrum/range?

Contents

- Thesis in a Nutshell
- Concepts: reflection & AOP
- Contributions
- Conclusions & Perspectives

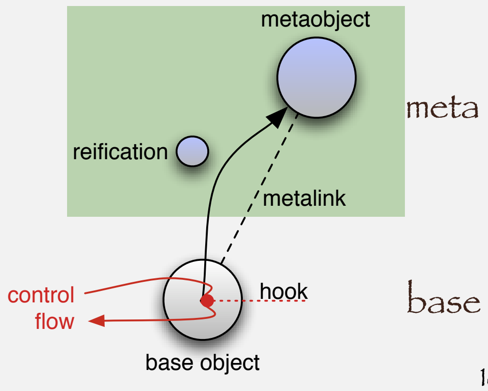
Contributions



- Partial Behavioral Reflection
- Versatile AOP Kernels
- Reflex/Java, Open Implementation
- Applications

Specific Context

- Behavioral reflection / runtime MOPs
  - metaobjects reasoning and acting upon reifications of a program described in terms of operations [McA96]
- Specific issues



1. Cost Issue

- Reifying operation occurrences is expensive

```
A a =
  o.foo(5);
```

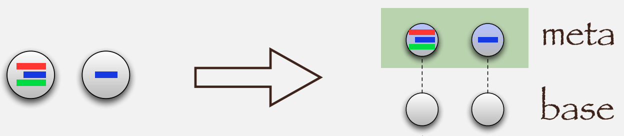
hook

```
A a =
  metaobj.handle(
    new Call(this, o,
      o.getClass().getMethod("foo"),
      new Object[]{new Integer(5)}));
```

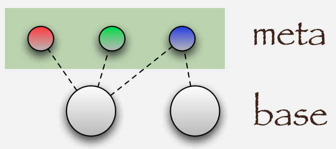
  - reify only when needed
  - reify only needed information (+how)

2. Metalink Issue

- Classical view on metalink
  - entity-based: per object, per class
    - leads to tangled metalevel



  - what we want: a concern-based metalevel



3. MOP Design Issue

- Definition of the precise protocol between levels
- Trade-off
  - expressiveness / performance / flexibility
- Frozen in existing reflective systems
  - at least rigid (e.g. [GowingCahill96])

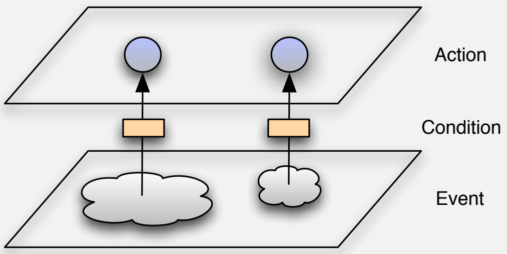
Proposal [OOPSLA03]

- Operational model of Partial Reflection
  - Selective reification
  - Links as configurable first-class entities
  - Open MOP support / MOP specialization

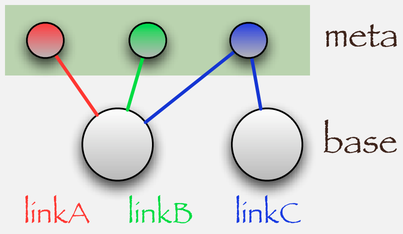
Selective reification

- Systematic analysis of partiality
- Spatial selection
  - what should be reified?
  - entities, operations, operation occurrences
- Temporal selection
  - when to reify?
  - dynamically-evaluated conditions
  - e.g. transparent futures

Links



- Flexible metalink
  - Group selected hooks in a hookset (1st class, composable)
  - Bind hookset to metaobject
  - Attributes
    - scope: global, class, object
    - activation: predicate
    - control: before, after, replace
    - ...



Open MOP Support

- Specific MOPs are defined by metalevel architects
  - what is an operation? which are supported?
  - interface of metaobjects (method and data)
- Several MOPs can coexist

MOP Specialization

- Flexible and fine-grained specialization
  - call generator descriptors
  - type, method and parameters
- MOP descriptors per [SCCC04]
  - operation
  - link
  - hookset

MOP Specialization for SOM

[ECOOP04]

```
somLink = API.links().addLink(MsgReceive.class, ...);
somLink.setControl(Control.BEFORE_AFTER);
```

Using standard MOP

```
_mo_somLink.beforeMsgReceive([m, r, args]);
..
_mo_somLink.afterMsgReceive([m, r, args, res]);
```

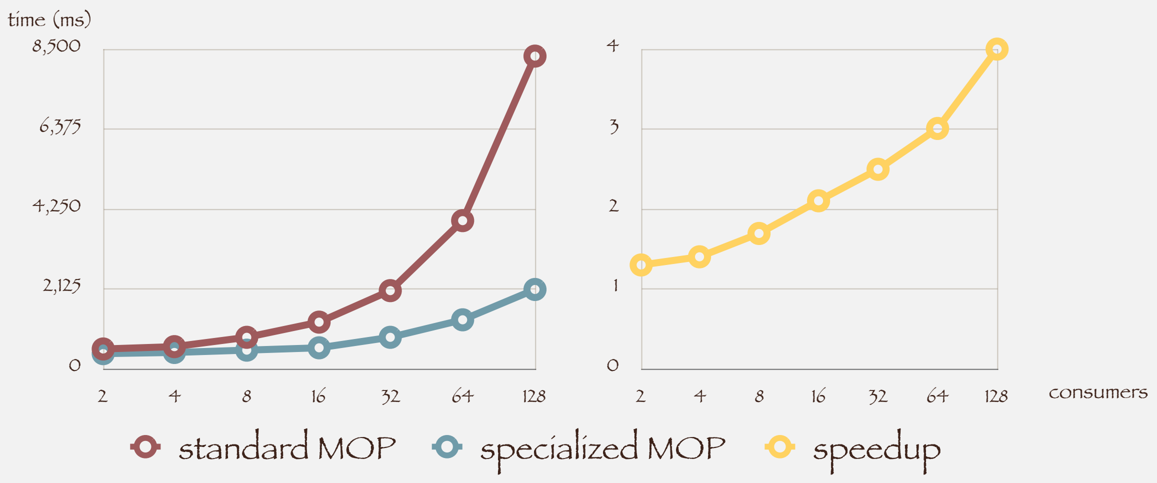
Using specialized MOP

```
somLink.setMOCall(Control.BEFORE, Scheduler.class, “enter”,
                    nameP, argsP);
somLink.setMOCall(Control.AFTER, Scheduler.class, “exit”);
```

```
_mo_somLink.enter(“put”, [ o ]);
..
_mo_somLink.exit();
```

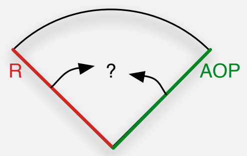
MOP Specialization for SOM

- buffer, 1 slot, 1 producer, n consumers -



as efficient as hand-made source code modification

Contributions



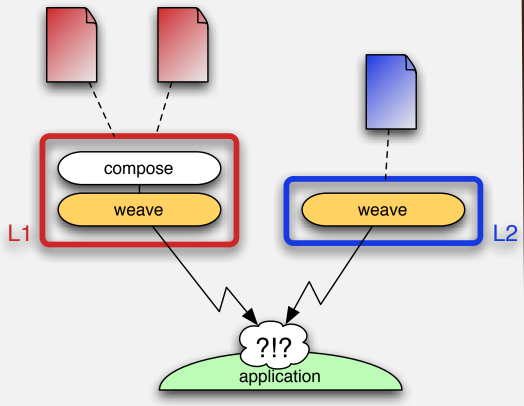
- Partial Behavioral Reflection
- Versatile AOP Kernels
- Reflex/Java, Open Implementation
- Applications

Specific Context

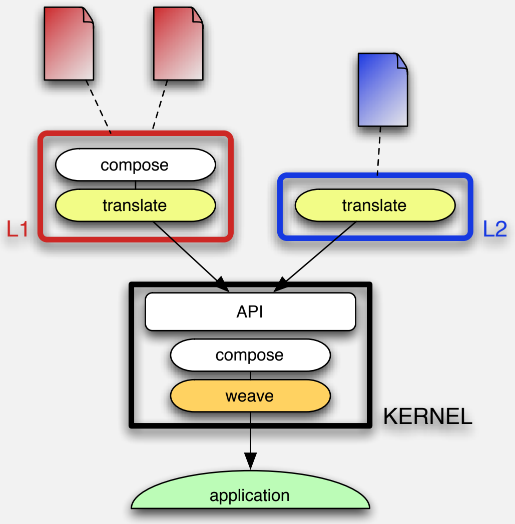
- Variety of AOP proposals
  - exploring the design space
    - different models
    - domain-specific vs. general-purpose
  - combining different approaches
    - depending on tackled concern (domain)
    - positive reports [Rashid01]

Issues

- Compatibility
- Reuse of weaver



AOP Kernels



Features of AOP

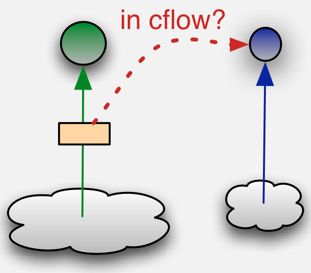
- Analysis of AOP proposals
  - asymmetric approaches (e.g., Pointcut-Advice) [Masuhara+03,Wand+04]
- Anatomy of AOP languages
  - sub-languages
    - cut: where
    - action: what
    - binding: association, instantiation
  - behavior and structure

Kernel Requirements [EIWAS04]

- Aspect languages
    - open support, modular integration
- Behavior and structure
    - expressive cut, complete action, separate binding
- Composition and collaboration
- Explicit interactions application/aspects
- Base-language compliance

Kernel Approach [(AOSD05)]

- Use partial reflection as base
    - generality + specializability ⇨ versatility
- Mapping
    - cut: introspection (hookset, activation)
    - action: intercession (metaobject)
    - binding: metalink (link, MOP)
- Abstraction gap
    - e.g., an AspectJ aspect with cflow



Kernel Requirements

- Aspect languages
  - open support, modular integration
- Behavior and structure
  - expressive cut, complete action, separate binding
- Composition and collaboration
- Explicit interactions application/aspects
- Base-language compliance

Aspect Languages

- Abstraction gap
  - 1 aspect = 1 linkset = n links
- Lightweight plugin architecture
  - plugin = AL parser + kernel definitions
  - AL general-purpose or domain-specific

```
schedule: BoundedBuffer with: MyScheduler
```

- Composition and traceability
  - conflicts detected on links
  - reported and resolved on linksets

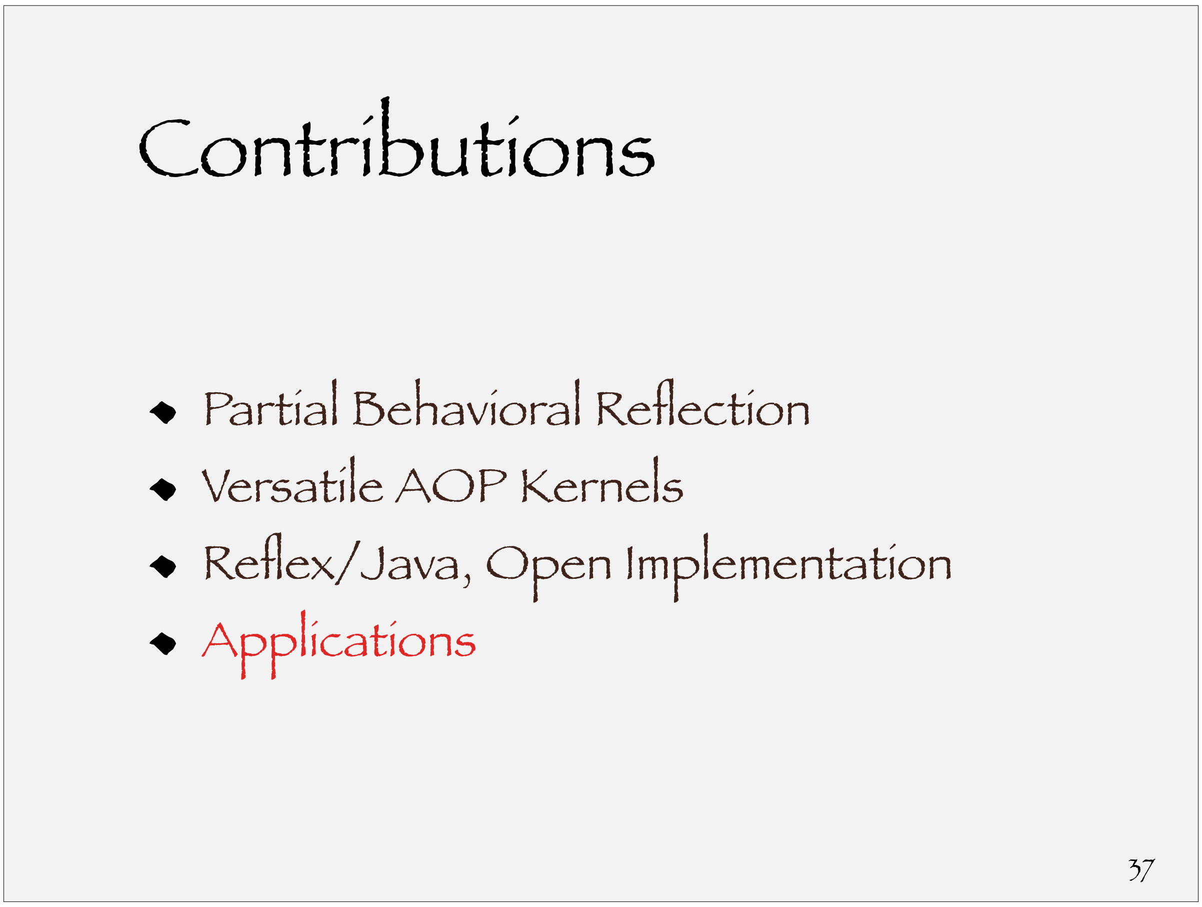
Contributions

- Partial Behavioral Reflection
- Versatile AOP Kernels
- Reflex/Java, Open Implementation
- Applications

Reflex for Java

[Reflection01,OOPSLA03,(AOSD05)]

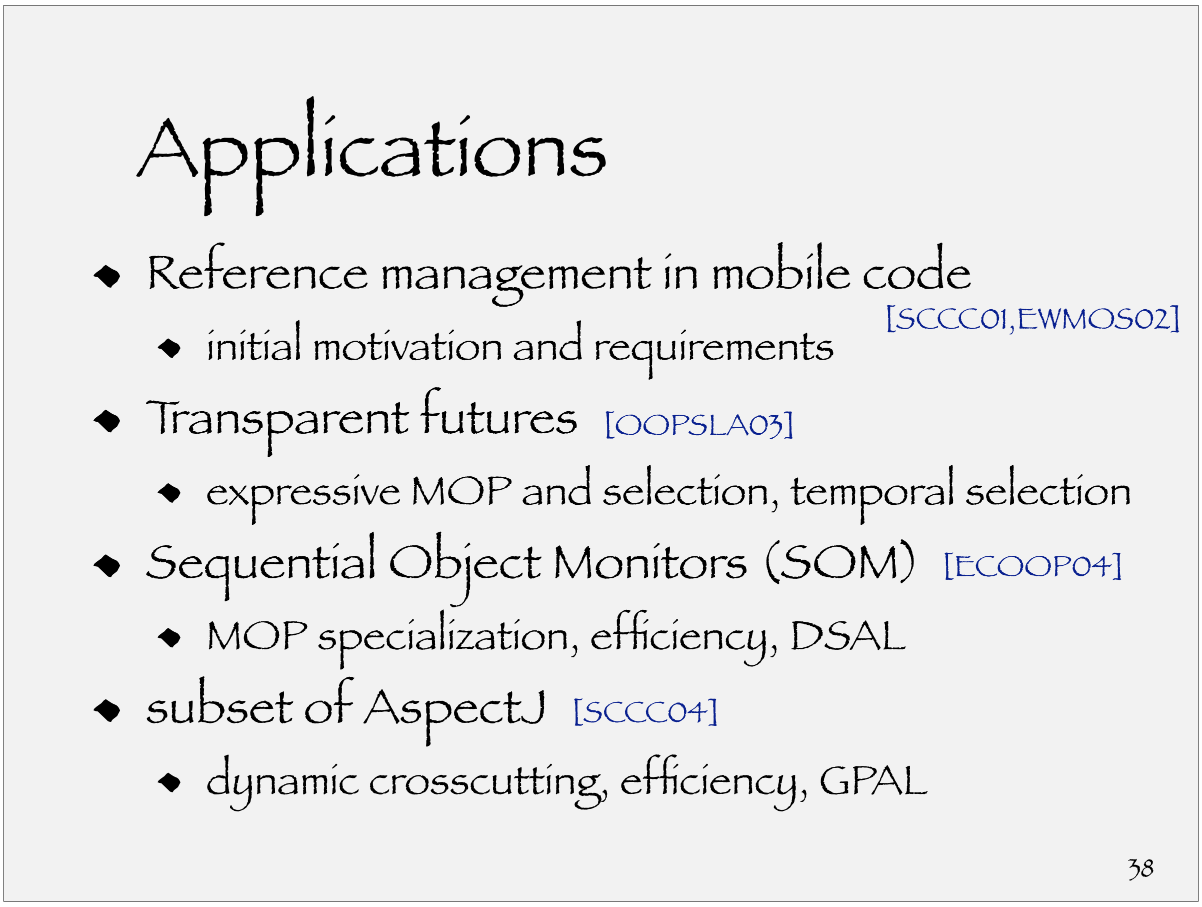
- Working implementation
  - portable and efficient
    - bytecode transformation (Javassist [Chiba00,Chiba+03])
- Open Implementation
  - iterative process, progressive decoupling
  - intensive use of OI design guidelines [Kiczales+97b]
  - modular and extensible:
    - Core Reflex / API (180 classes)
    - Standard library: operations, base metaobjects...
    - Tools, examples, plugins...



Contributions

- Partial Behavioral Reflection
- Versatile AOP Kernels
- Reflex/Java, Open Implementation
- Applications

37



Applications

- Reference management in mobile code
  - initial motivation and requirements [SCCC01,EWMOS02]
- Transparent futures [OOPSLA03]
  - expressive MOP and selection, temporal selection
- Sequential Object Monitors (SOM) [ECOOP04]
  - MOP specialization, efficiency, DSAL
- subset of AspectJ [SCCC04]
  - dynamic crosscutting, efficiency, GPAL

38

Contents

- Thesis in a Nutshell
- Concepts: reflection & AOP
- Contributions
- Conclusions & Perspectives

- Model of Partial Reflection
  - Achievements
    - balance trade-off between genericity/specificity
      - flexible metalink, MOP specialization
    - in between low-level and high-level tools
    - portable, efficient, applicable implementation
  - Perspectives
    - influence of “real-world constraints”
    - trade-off structure/adaptation
    - fully-static and fully-dynamic contexts

- AOP Kernels
  - Achievements
    - identification of the need and analysis
    - first prototype, including composition
    - combine power of reflection and guidance of aspect languages
  - Perspectives
    - AO models: prototypes, basic blocks
    - finer-grained, more precise, interactions
    - back to applications: Grid computing, Web apps, ...
      - DSALs: design, composition and interactions

Publications

with: Noury Bouraqadi, Denis Caromel (2), Pierre Cointe, Peter Ebraert (2), Luis Mateu, Jacques Noyé (6), José Piquer (3), Leonardo Rodríguez (2), Marc Ségura, Michael Vernaillen

Int. Conferences

- "A Versatile AOP Kernel for Java" (@AOSD'05)
- "Supporting Dynamic Crosscutting with Partial Behavioral Reflection: a Case Study" @SCCC'04
- "Sequential Object Monitors" @ECOOP'04
- "Partial Behavioral Reflection: Spatial and Temporal Selection of Reification" @OOPSLA'03
- "Altering Java Semantics via Bytecode Manipulation" @GPCE'02
- "Managing References upon Object Migration: Applying Separation of Concerns" @SCCC'01
- "Reflex - Towards an Open Reflective Extension of Java" @Reflection'01

Int. Workshops

- "Motivation and Requirements for a Versatile AOP Kernel" @EIWAS'04
- "A Concern-based Approach to Software Evolution" @DAW/AOSD'04
- "A Flexible Approach to Runtime Inspection" @ASARTI/ECOOP'03
- "Towards Transparent Adaptation of Migration Policies" @EWMOS/ECOOP'02
- "Runtime Metaobject Protocols: the Quest for their Holy Application" @PhDOOS/ECOOP'02

also: François Nollen, Angel Núñez, Guillaume Pothier, Rodolfo Toledo