Formal Reasoning on Component-Based Reconfigurable Applications

Nuno Gaspar Eric Madelaine Ludovic Henrio

Oasis Project Team
INRIA Sophia Antipolis - Méditerranée

40^{th} Symposium on Principles of Programming Languages - POPL'2013, January 25^{th}, 2013, Rome, Italy

Outline

Component-Based Engineering

- Software as *building blocks* that when put together form the intended functionality

- Core Elements
  - **Component**: Some entity, generally a piece of software code
  - **Interface**: Access point to/from components
  - **Binding**: Connection established between components, *through their interfaces*[1]

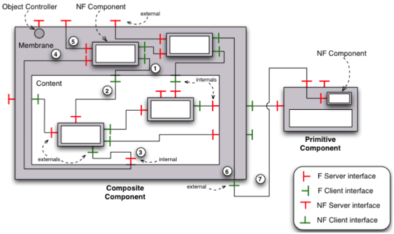
- Several components models proposed
  - CCM, CCA, SCA, Fractal, **GCM** (Grid Component Model), ...
  - Each with its intricacy
  - hierarchical/flat, distribution, reconfiguration, ...

[1] And this is the fundamental difference between Object-oriented programming and Component-based programming

Research Interests @ OASiS

Our work lies around the **GCM** Component Model

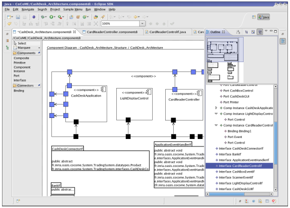
- Registered as a standard at the *European Telecommunications Standards Institute* (ETSI)
- A reference publication: *F. Baude et. al.* **GCM: a grid extension to Fractal for autonomous distributed components.** *Annals of Telecommunications, 64(1), 5-24, 2009* [3]
- And its behavioural model: *T. Barros et. al.* **Behavioural models for distributed Fractal components.** *Annals of Telecommunications, 64(1-2), jan 2009* [2]



Research Interests @ OASiS

Our work lies around the **GCM** Component Model

- Registered as a standard at the *European Telecommunications Standards Institute* (ETSI)
- Complete Software Life Cycle
  - **VerCors** - A Specification & Verification Platform for GCM Applications
  - **GCM/ProActive** - A Java Library



Research Interests @ OASiS

Our work lies around the **GCM** Component Model

- Registered as a standard at the *European Telecommunications Standards Institute* (ETSI)
- Complete Software Life Cycle
  - **VerCors** - A Specification & Verification Platform for GCM Applications
  - **GCM/ProActive** - A Java Library
- Research Projects & Industrial Partners
  - Spinnaker, PLAY, CompatibleOne, ...
  - Tagsys, Renault, Orange, ...

Approach to Verification

- Typically, our projects requires us to build some **GCM** application
- For which we prove the intended properties via the **CADP Model-Checker**
  - See for instance [1, 4, 6]
- Often, we need Distributed Space-State Generation...
  - Tackle the need for huge space-state generation by
    - abstraction
    - compositional and contextual reduction
    - distributed generation

Approach to Verification

- Typically, our projects requires us to build some **GCM** application
- For which we prove the intended properties via the **CADP Model-Checker**
- Often, we need Distributed Space-State Generation...
  - Still, we face the common **space-state explosion** phenomena
- Specifying → (distributed) Space-state Generation → System Product → Model-Checking
  - Not that simple in practice...
  - Need access to a Grid/Cloud environment
    - Better be friendly to the SysAdmin too :-)
  - Space-state generation takes time (upto several days)
  - Specification is rarely right at first shot
  - Constrained by the use of finite domains

Mefresa in a Nutshell

- A Mechanized Framework for Reasoning on Software Architectures
- Gives a Formal Semantics to the **GCM Component Model**
- Developed with the **Coq Proof Assistant**

http://www-sop.inria.fr/members/Nuno.Gaspar/Mefresa.php

Mefresa in a Nutshell - Aims 1/3

- A Mechanized Framework for Reasoning on Software Architectures
- Gives a Formal Semantics to the **GCM Component Model**
- Developed with the **Coq Proof Assistant**

Three types of aims

1. Disambiguate the informal Specification
   - Allows us to prove properties expected to hold
   - e.g. *"... ensure that primitive bindings* **cannot cross** *component boundaries except through interfaces."*

   Within our Framework, it boiled down to:

```
Theorem cross_binding_cannot_happen:
  forall b system,
    well_formed system      ->
    system_binding b system ->
    cross_binding b system  ->
    False.
Proof.
```

Mefresa in a Nutshell - Aims 2/3

- A Mechanized Framework for Reasoning on Software Architectures
- Gives a Formal Semantics to the **GCM Component Model**
- Developed with the **Coq Proof Assistant**

Three types of aims

1. Disambiguate the informal Specification
   - Allows us to prove properties expected to hold

2. Proof of general algorithms manipulating GCM Applications
   - e.g. L. Henrio, M. Rivera. **Stopping safely hierarchical distributed components: application to GCM**, *ACM CBHPC '08* (see [7])

Mefresa in a Nutshell - Aims 3/3

- A Mechanized Framework for Reasoning on Software Architectures
- Gives a Formal Semantics to the **GCM Component Model**
- Developed with the **Coq Proof Assistant**

Three types of aims

1. Disambiguate the informal Specification
   - Allows us to prove properties expected to hold

2. Proof of general algorithms manipulating GCM Applications

3. Proving that some GCM application meets the specification
   - Purely structural concerns:
     - Reconfiguration X leads us to a **well formed** state
   - Functional concerns: encode a model-checker inside Coq (see [8])
     - Scalability may be an issue here...
   - Take Model-Checking results as assumptions

Mefresa in a Nutshell - Approach 1/4

Approach

- Encoding of main **GCM** elements

```
Inductive component : Type :=
  | Component    : ident          -> type          ->
                   path           -> controlLevel  ->
                   list component -> list interface ->
                   list binding   -> component.
```

implicitly models the GCM Hierarchical structure

and in the same spirit for **Interface** and **Binding**..

Mefresa in a Nutshell - Approach 2/4

Approach

- Encoding of main **GCM** elements
- A simple *operation* language

$$\begin{array}{lll} \text{op} & ::= & \textbf{mk\_component}\ \text{component} \\ & | & \textbf{mk\_interface}\ \text{interface} \\ & | & \textbf{mk\_binding}\ \text{binding} \\ & | & \textbf{rm\_component}\ \text{component} \\ & | & \textbf{rm\_binding}\ \text{binding} \\ & | & \text{op; op} \\ & | & \textbf{done} \end{array}$$

Design (and reconfiguration) of software architectures seen as **transitions**:

$$\longrightarrow : (operation \times state) \rightarrow (operation \times state) \rightarrow Prop$$

$$\text{e.g. } \langle op, s \rangle \longrightarrow \langle op', s' \rangle$$

Mefresa in a Nutshell - Approach 2/4

Approach

- Encoding of main **GCM** elements
- A simple *operation* language
  - To which we attach a proof rule to each constructor:

$$\frac{\begin{array}{l} c = Component\ id\ t\ p\ cl\ subComps\ interfaces\ bindings \\ valid\_path\ p\ s \\ well\_formed\_components\ subComps \\ well\_formed\_interfaces\ interfaces \\ \forall\ id', id' \in (get\_scope\ p\ s) \rightarrow (id \neq id') \end{array}}{\langle make\_component\ c, s\rangle \longrightarrow \langle done, (add\_component\ s\ c)\rangle}$$

and in the same spirit for the remaining constructors...

Mefresa in a Nutshell - Approach 3/4

Approach

- Encoding of main **GCM** elements
- A simple *operation* language
- A proof of correction for our semantic rules

  for all operations starting in a **well formed** state, upon completion, we end up in a **well formed** state

```
Theorem validity:
  forall op s s',
   well_formed s              ->
   op / s --->* Done / s'     ->
   well_formed s'.
```

by other words, using our semantic rules **ensure** that you produce GCM architectures that meet the specification

Mefresa in a Nutshell - Approach 4/4

Approach

- Encoding of main **GCM** elements
- A simple *operation* language
- A proof of correction for our semantic rules
- Co-Induction to model communication between components
  - Infinite traces

Final Remarks

We Have

- A Semantics for the **GCM** Component Model Mechanized in Coq
- Our operation language **proved** correct w.r.t to the specification
    - Building & Reconfiguration of GCM architectures *correct-by-construction*
- Preliminary experiments regarding **proved reconfiguration scripts** & communication modelling via **co-inductive** predicates

We Want

- We aim at providing an unified framework for reasoning on **structural** concerns, **functional** concerns and their interaction
- Seamless integration with VerCors [4]
    - At the 3 types of aims discussed before
- Understand fully to what extend our Verification methodology (with Model Checking) can benefit from Mefresa

Hope you liked it!

Thank you for listening!

email me your thoughts at

(fun x y ⇒ x . y @inria.fr) Nuno Gaspar

Raba Ameur-boulifa, Ludovic Henrio, and Eric Madelaine.
Behavioural models for group communications.
In *In in proceedings of the International Workshop on Component and Service Interoperability, WICS10, Malaga*, 2010.

Tomás Barros, Rabéa Ameur-Boulifa, Antonio Cansado, Ludovic Henrio, and Eric Madelaine.
Behavioural models for distributed fractal components.
Annales des Télécommunications, 64(1-2):25–43, 2009.

Françoise Baude, Denis Caromel, Cédric Dalmasso, Marco Danelutto, Vladimir Getov, Ludovic Henrio, and Christian Pérez.
Gcm: a grid extension to fractal for autonomous distributed components.
Annales des Télécommunications, 64(1-2):5–24, 2009.

Raba Ameur Boulifa, Raluca Halalai, Ludovic Henrio, and Eric Madelaine.

Verifying safety of fault-tolerant distributed components.
In *International Symposium on Formal Aspects of Component Software (FACS 2011)*, Lecture Notes in Computer Science, Oslo, 2011. Springer.

Eric Bruneton, Thierry Coupaye, and Jean-Bernard Stefani.
The fractal component model, 2004.

Antonio Cansado and Eric Madelaine.
Formal methods for components and objects.
chapter Specification and Verification for Grid Component-Based Applications: From Models to Tools, pages 180–203. Springer-Verlag, Berlin, Heidelberg, 2009.

Ludovic Henrio and Marcela Rivera.
Stopping safely hierarchical distributed components: application to gcm.

In *CBHPC '08: Proceedings of the 2008 compFrame/HPC-GECO workshop on Component based high performance*, pages 1–11, New York, NY, USA, 2008. ACM.

Christoph Sprenger.
A verified model checker for the modal -calculus in coq.
In *In TACAS, volume 1384 of LNCS*. Springer Verlag, 1998.