Mastering Complex Systems with Heterogeneous Components

Christian ATTIOGBE

AeLoS / LS2N – UMR CNRS 6004

Séminaire interne AeLoS - 09/02/2017

LS2N LABORATOIRE DES SCIENCES DU NUMÉRIQUE DE NANTES

Plan

Plan

Context

Context : Component-based Software Engineering (CBSE)

- Correction-by-construction of Software, by assembling components
- Combining third party components, from different providers/developers
- Distributed design (compared with Object-oriented soft. eng, centralised design).
- Partial views of the models, the design, the formal analysis,
- Heterogeneity
- Continuous impact of the environment on execution, reliability, maintenance

Contract : once upon a time !

Specification

A Specification is a contract between an user and a software (more generally a system). - what the system *should* do

- FLOYD-HOARE, 1969, fundamental *laws*
- Program correction
- Formal Reasoning
- Rigorous programming (pre-post)

context

Sequential software
Design by Contract, MEYER

Concurrency & contract : once upon a time !

,

- OWICKI, GRIES (1976), extension of Hoare'*laws* shared variables
- C. JONES, (1981) *Rely-Guarantee*, O.G.+compositional
- System Design and Formal Reasoning (safety, lifeness) (**Assume-Guarantee**)

context

Concurrent programs/software
Compositional reasoning principle
ABADI & LAMPORT(1993)

Compositional principle :
(Pnueli 1985), (Clarke 1989), ..., (Abadi&Lamport 1993, 1995)

Rely Guarantee

P a program
$(Pre, rel, guar, post)$ the specification of P

A program P satisfies its specification if

assumption under the assumptions that

- P is started in a state that satisfies *Pre*
- any environment transition in the computation of P satisfies *rely*

Commitments its ensures

- any component transition satisfies *guar*
- if P terminates, the final state satisfies *Post*

Further developments

- Mechanization (with various provers)
- Specification theories
- Soundness, Completness : various theories
- I/O automata (N. LYNCH, 1987) (L. DE ALFARO, T. HENZINGER, 2001)
- Algebraic view, State-based view,
- AG framework for I/O automata (K. LARSEN, 2006)

never give up !

Models and programs should be analysed.

Operating systems should work.

Distributed Operating systems, Distributed applications and databases.

CBSE : during that time !

- JavaBeans !
- Debug, debug, debug, debug
- Heavy cost of failures, maintenance
- Is that rigorous ? (correct ?) Rigorous design

Emergency

Contracts ! contracts ! contracts !

Funding international projects

Component Models

Models	Syntax	Semantics
JavaBeans, EJB	Object-oriented prog languages	Classes
COM, .NET, CCM, Fractal, Web Services	Prog languages with IDL mappings	Objects
ACME-like ADLs, UML2.0, Kobra, Koala, SOFA, PE-COS	modeling language, ADL	Architectural Units
LOTOS, BIP	modeling language, ADL	LTS interleaving
Kmelia	modeling language, ADL	LTS interleaving
...	...	...
		I/O automata, traces

(Software component models, Kung-Kiu Lau and Zheng Wang, IEEE TSE vol 33, october 2007)

Plan

Recent European/International Initiatives

Contracts for Systems Design : Theory

Albert Benveniste, Benoît Caillaud, Dejan Nickovic, Roberto Passerone,
Jean-Baptiste Raclet, Philipp Reinkemeier, Alberto Sangiovanni-Vincentelli,
Werner Damm, Tom Henzinger, Kim G. Larsen
Project-Teams Hycomes

- A Meta-Theory for component-based design.
- Several instanciations : for Real-Time, SysML, AADL,...

Meta-Theory : components

A component

$$M_1 : \begin{cases} \text{variables:} & \begin{cases} \text{inputs:} & x, y \\ \text{outputs:} & z \end{cases} \\ \text{types:} & x, y, z \in \mathbb{R} \\ \text{behaviors:} & (y \neq 0 \rightarrow z = x/y) \wedge (y = 0 \rightarrow z = 0) \end{cases}$$

A *contract*, denoted by the symbol $\mathscr{C}$, is a way of specifying components with the following characteristic properties:

1. Contracts are intentionally abstract;
2. Contracts distinguish responsibilities of a component from those of its environment.

Meta-Theory : contracts

A contract

$$\mathscr{C}_1 : \begin{cases} \text{variables:} & \begin{cases} \text{inputs:} & x, y \\ \text{outputs:} & z \end{cases} \\ \text{types:} & x, y, z \in \mathbb{R} \\ \text{assumptions:} & y \neq 0 \\ \text{guarantees:} & z = x/y \end{cases}$$

$\mathscr{C}_1$ defines the set of components having as variables {inputs: x, y; output: z} of type real, and whose behaviors satisfy the implication

"assumptions $\Rightarrow$ guarantees"

Meta-Theory : concepts

Concept	Definition and generic properties	What depends on the particular theory of contracts
Primitive		
Component	Components are denoted by M; they can be *open* or *closed*	How components are specified
Composability of components	A type property on pairs of components (M_1, M_2)	How this type property is defined
Composition of components	$M_1 \times M_2$ is well defined if and only if M_1 and M_2 are composable; It is required that $\times$ is associative and commutative	The definition of the composition
Environment	An *environment* for component M is a component E such that $E \times M$ is defined and closed	
Derived		
		Which family $\mathbf{C}$ of contracts

Meta-Theory : derived concepts

Derived		
Contract	A *contract* is a pair $\mathcal{C} = (\mathcal{E}_\mathcal{C}, \mathcal{M}_\mathcal{C})$, where $\mathcal{M}_\mathcal{C}$ is a subset of components and $\mathcal{E}_\mathcal{C}$ a subset of legal environments	Which family $\mathbf{C}$ of contracts can be expressed, and how they are expressed; unless otherwise specified, quantifying is implicitly over $\mathcal{C} \in \mathbf{C}$
Consistency	$\mathcal{C}$ is *consistent* iff it has at least one component: $\mathcal{M}_\mathcal{C} \neq \emptyset$	How consistency is checked
Compatibility	$\mathcal{C}$ is *compatible* iff it has at least one environment: $\mathcal{E}_\mathcal{C} \neq \emptyset$	How compatibility is checked
Implementation	$M \models^{\mathrm{M}} \mathcal{C}$ if and only if $M \in \mathcal{M}_\mathcal{C}$ $E \models^{\mathrm{E}} \mathcal{C}$ if and only if $E \in \mathcal{E}_\mathcal{C}$	How implementation is checked
Refinement	$\mathcal{C}' \preceq \mathcal{C}$ iff $\mathcal{E}_{\mathcal{C}'} \supseteq \mathcal{E}_\mathcal{C}$ and $\mathcal{M}_{\mathcal{C}'} \subseteq \mathcal{M}_\mathcal{C}$; Property 1 holds	How refinement is checked
GLB and LUB of contracts	$\mathcal{C}_1 \wedge \mathcal{C}_2 =$ Greatest Lower Bound (GLB) for $\preceq$ we assume GLB exist $\mathcal{C}_1 \vee \mathcal{C}_2 =$ Least Upper Bound (LUB) for $\preceq$ we assume LUB exist Property 2 holds	Whether and how GLB and LUB can be expressed and computed
Composition of contracts	$\mathcal{C}_1 \otimes \mathcal{C}_2$ is defined if $\left.\begin{array}{l} M_1 \models^{\mathrm{M}} \mathcal{C}_1 \\ M_2 \models^{\mathrm{M}} \mathcal{C}_2 \end{array}\right\} \Rightarrow (M_1, M_2)$ composable $\mathcal{C}_1 \otimes \mathcal{C}_2 = \bigwedge \left\{ \mathcal{C} \left\vert \begin{array}{c} M_1 \models^{\mathrm{M}} \mathcal{C}_1 \text{ and } M_2 \models^{\mathrm{M}} \mathcal{C}_2 \text{ and } E \models^{\mathrm{E}} \mathcal{C} \\ \Downarrow \\ M_1 \times M_2 \models^{\mathrm{M}} \mathcal{C} \text{ and } E \times M_2 \models^{\mathrm{E}} \mathcal{C}_1 \text{ and } E \times M_1 \models^{\mathrm{E}} \mathcal{C}_2 \end{array} \right. \right\}$ Assumption 1 is in force; Properties 3, 4, and 5 hold Say that $\mathcal{C}_1$ and $\mathcal{C}_2$ are *compatible* if so is $\mathcal{C}_1 \otimes \mathcal{C}_2$	How composition is expressed and computed
Quotient	$\mathcal{C}_1 / \mathcal{C}_2 = \bigvee \{ \mathcal{C} \mid \mathcal{C} \otimes \mathcal{C}_2 \preceq \mathcal{C}_1 \}$; Property 6 holds	How quotient is expressed and computed

Table IV
Summary of the meta-theory of contracts. We first list **primitive** concepts and then **derived** concepts introduced by the meta-theory.

Plan

A Theory of Heterogeneous Components with generalised contracts

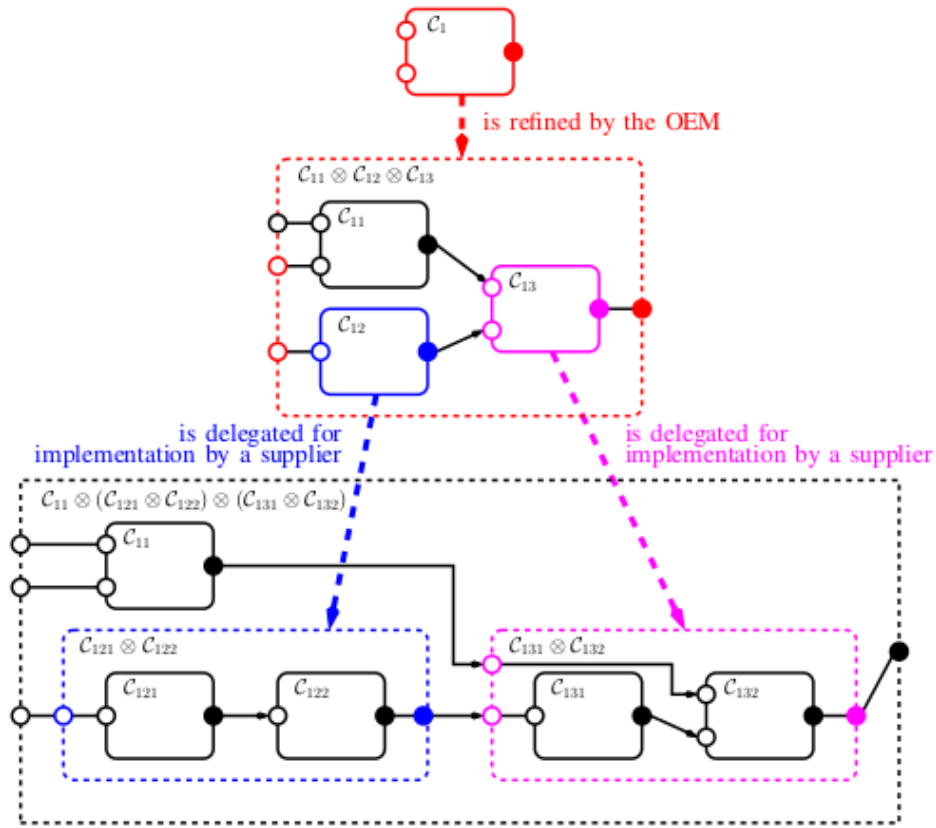

Figure 4. Stepwise refinement

(Contract for system design, SCP, 2013)

A Theory of Heterogeneous Components with generalised contracts

Challenges

A modelling framework for heterogeneous components together with a compositional (specification/contract) theory for reasoning about safety, progress, non-fuctional properties of components and systems.
Compositionality, Refinement (specifications/models to code), Substituvity, reliability,...

Proposal :

$$\{P_i\}, \{P_j\}, \{P_k\}, \ldots \ \mathbf{S} \ \{Q_u\}, \{Q_v\}, \ldots$$

Challenges

- Define a multi-level proof system which considers the combination of contracts and combine appropriate reasoning/proof techniques/tools to overcome consistency apsects and compositionality.
- consistency between contracts (between the layers) if necessary and meaningful
- A layered structure where the layers are labelled as suggested, to enforce heterogeneity and interoperability
- Global analysis : one specific facet can be considered ; several facets can be considered ; all facets can be considered.
- A projection on one aspect may simply result in an usual contract-based reasoning

A Theory of Heterogeneous Components with Generalised Contracts

Définition (Generalised contracts)

A generalised contract = a multi-facets contract
a facet deals with one specific aspect : typing, behavior
functional : safety and liveness,
non-functional : time, QoS, energy, memory, cpu, performance, ...

Ingredients : generalised contract

+++	P^{+++}
Perf	P^{perf}
QoS	P^{qos}
T	P_j^T
L	P_j^L
Fun	P_i^F

Layering properties + projection

$\{P_i^F\}, \{P_j^L\}, \{P_k^T\}, \ldots$ **M** $\{Q_u^F\}, \{Q_v^L\}, \ldots$

Projection on input languages of dedicated provers

Composition of the properties

Feedback of analysis on modelling

Ingredients

- Meta-Theory..., Benveniste & Al
- Multi-level contracts (Kmelia Serv. compo, ass, WCIS'10, Amaretto'17),
- BIP (compositionality results), Lotos/CADP (LNT, GRL)
- Extension of the Property Specification Language (PSL) to deal with multiple properties (μPSL)
- Bridging μPSL with existing provers
- Linking μPSL with component modelling languages
- Focus on embedded systems (AFSEC, ...)
- ...

Related Works

Meta-Theory : Component System Design (Benvesite & Al)
Ptolemy II : Ed. Lee
C. Chilton, B. Jonsson, M. Kwiatkowska, FACS2012
C. Chilton, *An algebraic theory of componentised interaction,* Ph.D. thesis, Department of Computer Science, University of Oxford, 2013.
. . .

Discussions

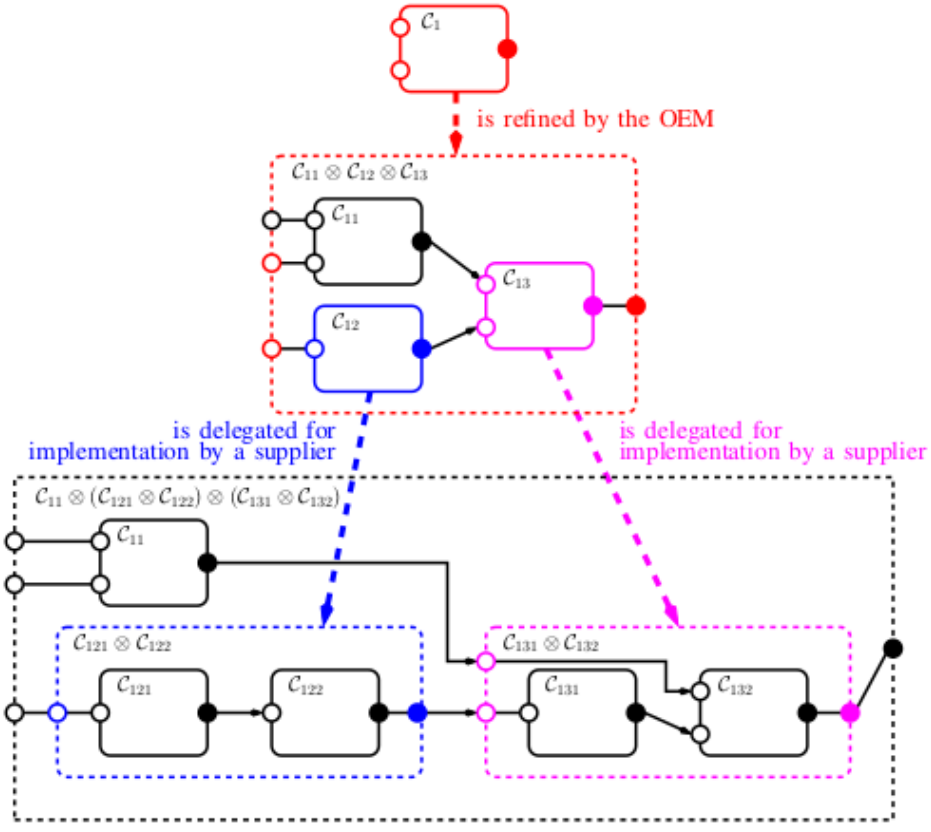

Figure 4. Stepwise refinement

discussions (completed after the talk)

Some remarks from colleagues :

- interesting ! BUT ...
- Contracts are too abstract (compared with entities used by programmaers)
- Many (hard) theoretical works are needed to overcome the issues
- Need focus on more pragmatical works
- why not digging into engineering ?
- components are not really used by every day developpers

Other terms of the discussions : we should compose to build (well) software ! contracts are only one way to keep abstraction of the environment.