Multiple Viewpoint Contract-Based Specification and Design

Benveniste et al.

Discussion session – EE 249
Huy Vo and Behrooz Shahsavari

Overview

- Mathematical foundations and the design methodology of the contract-based model developed in the framework of the SPEEDS project
- SPEEDS aims at developing methods and tools to support “speculative design”
- This is achieved by
  - focusing on behaviors
  - supporting the notion of “rich component”
  - representing rich components via their set of associated contracts
  - formalizing the whole process of component composition

MODEL OVERVIEW

Model overview

- Development of a model that supports concurrent system development
- Ability to support abstraction mechanisms
- Work with multiple viewpoints
- Model should not force a specific model of computation and communication

Model overview

- Objective:
  - Develop a theory and methodology of component based development
- Two broad families of approaches:
  - Building systems from library components
    - details may be omitted
    - mechanism of abstraction
  - Distributed systems development with highly distributed supplier chains
    - Splitting and distributing responsibilities
    - Interaction

Model overview

- Each supplier is given a design task
  - A goal, also called *guarantee* or *promise*

- Other entities, which are not under the responsibility of this supplier:
  - Assumptions
  - Achieving its own promises
  - *Contracts*

Components and Contracts

- A component is a hierarchical entity that represents a unit of design
- Components are connected together to form a system
- A component may include both *implementations* and *contracts*

Components and Contracts

- notion of a contract for a component:
  - a pair of assertions

- A contract is an assertion on the behaviors of a component (the promise) subject to certain assumptions
  - contract *C* as a pair (*A*,*G*)

Components and Contracts

- An implementation of a component satisfies a contract whenever it satisfies its promise, subject to the assumption

Components and Contracts

Definition 1 (Parallel Composition). *Let* $C_1 = (A_1, G_1)$ *and* $C_2 = (A_2, G_2)$ *be contracts. The parallel composition* $C = (A, G) = C_1 \parallel C_2$ *is given by*

$$A = (A_1 \cap A_2) \cup \neg(G_1 \cap G_2), \tag{1}$$

$$G = G_1 \cap G_2, \tag{2}$$

Definition 2 (Elimination). *For a contract* $C = (A, G)$ *and a port* p*, the* elimination of p in C *is given by*

$$[C]_p = (\forall p\, A,\ \exists p\, G) \tag{3}$$

where A *and* G *are seen as predicates.*

Definition 3 (Dominance). *We say that* $C = (A, G)$ dominates $C' = (A', G')$*, written* $C \preceq C'$*, if and only if*

$$A \supseteq A', \text{and}$$

$$G \subseteq G',$$

and the contracts have the same ports.

Components and Contracts

Definition 4 (Bounds). *For contracts* $C_1 = (A_1, G_1)$ *and* $C_2 = (A_2, G_2)$ *(in canonical form), we have*

$$C_1 \sqcap C_2 = (A_1 \cup A_2, G_1 \cap G_2), \tag{4}$$

$$C_1 \sqcup C_2 = (A_1 \cap A_2, G_1 \cup G_2). \tag{5}$$

System Obligations

- Contracts are not the only way a designer would like to express system's requirements.
  - System obligations are typically high level requirements

- System obligations should be checked on contracts as early as possible

Asymmetric Role of Ports

- Contracts are not the only way a designer would like to express system's requirements.
  - System obligations are typically high level requirements
- System obligations should be checked on contracts as early as possible
- Visible and Local ports

Fusion

- Combining conjunction and parallel composition may not be trivial
- Fusion considers only compositions of contracts for which
  - internal connections have been fully established
  - and discharged

Methodology

- Now that relations and operations on contracts are defined…
- Can use these relations in the design process
  - Define possible design steps
  - Verify those design steps

Elements of Design

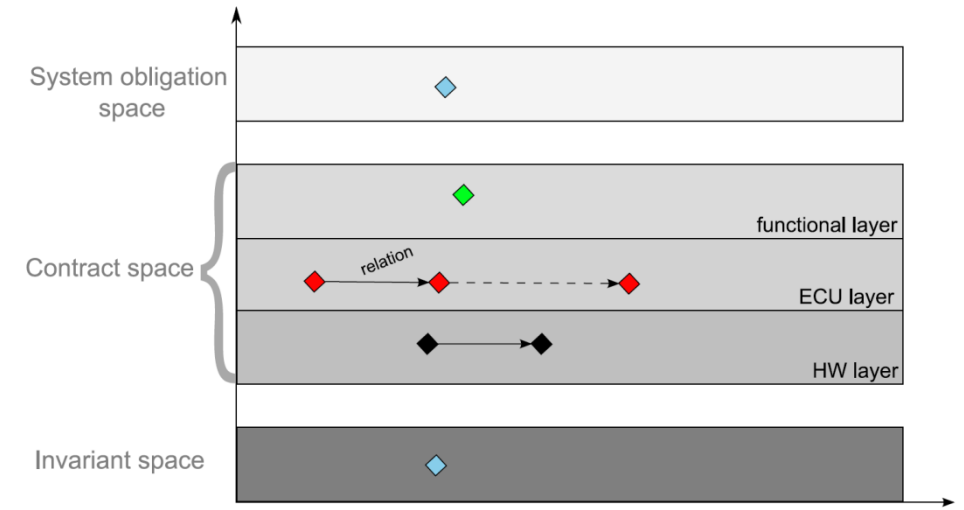

Fig. 3. Organization of spaces and layers

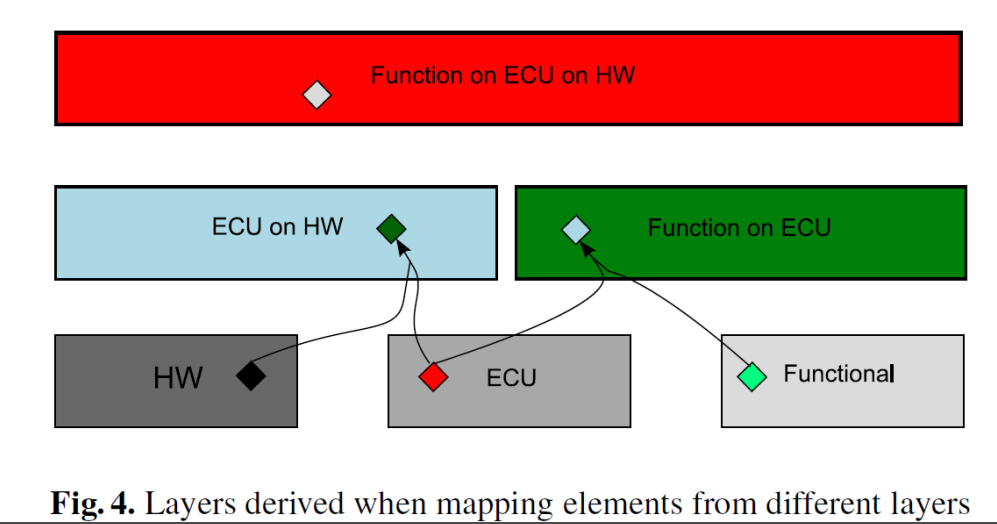

Fig. 4. Layers derived when mapping elements from different layers

- Design elements organized into 3 design spaces and layers
- Design elements are connected with relation arrows
- Mapping elements from one layer to another layer creates an element in a new layer

Design Step

- A design step is an evolution of the development of the design
  - Defined as a tuple of *source* and *target* rich components
  - Has “required” relations between the elements of the tuple
  - A design step is “validated” if all the required relations hold

Quick sidebar

- RC = {B, C, M} source rich component
- RC’ = {B’, C’, M’} target rich component
- B,B’ are the system obligations
- C,C’ are the contracts
- M,M’ the implementation

Design Steps on Single Rich Components

- **System obligation modification design step**
  - C = C', M = M', B=/=B'
  - Verify that $B' \subseteq B$ or C' conforms to $B \cup \neg B'$
- **Contract modification design step**
  - B = B', M = M', C =/= C'
  - Verify that C' strongly dominates C or C' is compatible, consistent, conforms to C and M' satisfies it
- **Implementation modification design step**
  - B = B', C = C', M=/=M'
  - Verify that M' refines M or that M' satisfies C

Design Steps on Multiple Rich Components

- **Decomposition design step**
  - 1 source, 2 or more targets
  - Verify that the source contract (C'') strongly dominates the first target contract (C_1)
  - Or that C'' is compatible, consistent and conforms to (C_1) and M'' satisfies C_1
- **Composition design step**
  - Multiple sources and 1 target
  - No verification since parallel composition preserves strong dominance, satisfaction and refinement