What makes the Difference? - Basic Characteristics of Configuration

Lothar Hotz
HITeC e.V., University of Hamburg
hotz@informatik.uni-hamburg.de

Configuration Model (*CM*)

Component description

```
(Entity-Model    name: Menu
                 super-type: Aggregate
                 data-properties:
                        ((kindOfTaste {hearty light}))
                 hasCourses:
                        {(AntiPasti :min 0 :max 1)
                         (MainCourse :min 1 :max 1)
                         (Dessert :min 0 :max 1)})

(Entity-Model    name: AntiPasti
                 super-type: Part)

(Entity-Model    name: MainCourse
                 super-type: Part)

(Entity-Model    name: Dessert
                 super-type: Part)

(Constraint      name: HeartyEqualsToAntiPasti
                 Menu.kindOfTaste == hearty <=>
                              Menu.hasCourses HAS (AntiPasti :min 1 :max 1))
```

Taxonomical relation

Data-type property

Structural property

Structural property domain

N-ary relation

Requirements

```
(Entity-Instance name: menu-1
                 entity-model: Menu
                 kindOfTaste: {hearty}
                 hasCourses: {mainCourse-1})
```

Resulting Configuration

Some reasoning,
e.g., constraint processing

```
(Entity-Instance name: mainCourse-1
                 entity-model: MainCourse)
(Entity-Instance name: menu-1
                 entity-model: Menu
                 kindOfTaste: {hearty}
                 hasCourses: {mainCourse-1 antiPasti-1})

(Entity-Instance name: mainCourse-1
                 entity-model: MainCourse)
(Entity-Instance name: antiPasti-1
                 entity-model: AntiPasti)
```

inferred

No dessert!

Need for Incremental Configuration

- Under-specified problems are a usual case in configuration.
  - E.g., no decision about dessert by the user
- How to select one solution if alternative solutions exist?
- Often, users understand their requirements only during the course of configuration.

Property Domain

Partial property domain

```
hasCourses: {mainCourse-1
             (AntiPasti :min 0 :max 1)
             (Dessert :min 0 :max 1)}
```

Terminal property domain

```
kindOfTaste: {hearty [terminal]}
```

Configuration requirements:

Set of instances with some property values set to terminal property domain

```
{(Entity-Instance name: menu-1
                  entity-model: Menu
                  kindOfTaste: {hearty} [terminal]}
                  hasCourses: {mainCourse-1}) … }
```

Partial Configuration and Final Configuration

Partially filled entity instance

```
(Entity-Instance name: menu-1
                 entity-model: Menu
                 kindOfTaste: hearty
                 hasCourses: {mainCourse-1
                              (AntiPasti :min 0 :max 1)
                              (Dessert :min 0 :max 1)})
```

Completely filled entity instance

```
(Entity-Instance name: menu-1
                 entity-model: Menu
                 kindOfTaste: {hearty [terminal]}
                 hasCourses: {mainCourse-1 antiPasti-1 [terminal]})
```

Partial configuration (*PC*): Set of partially or completely filled entity instances.

Final configuration: Set of completely filled entity instances.

Variable

Variable (*V*)

```
(Variable
   entity-instance: menu-1
   property: hasCourses
   property-value: {mainCourse-1
                    (AntiPasti :min 0 :max 1)
                    (Dessert :min 0 :max 1)})
```

Represents one decision of setting the value of one property.
The variable domain represents possible outcomes of the decision as a property value.
In the course of configuration, for one property there might be several variables.

Open Issue Operator (*OIO*)

Partial Configuration (*PC*)

```
{(Entity-Instance
   name: menu-1
   entity-model: Menu
   kindOfTaste: {hearty}
   hasCourses: {mainCourse-1})
…}
```

Configuration Model (*CM*)

```
(Entity-Model name: Menu
   super-type: Aggregate
   data-properties:
    ((kindOfTaste {hearty light})
   hasCourses: {(AntiPasti :min 0 :max 1)
                (MainCourse :min 1 :max 1)
                (Dessert :min 0 :max 1)})
```

Variables collected in an "what to do" agenda (*A*)

OIO: CM, PC → $V_i \in A$

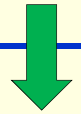
```
{(Variable
   entity-instance: menu-1
   property: hasCourses
   property-value: {mainCourse-1
                    (AntiPasti :min 0 :max 1)
                    (Dessert :min 0 :max 1)})
…}
```

OIO computes new variables, i.e., decision to be made.
Also for new entity instances!

Select Operator (*SO*)

Agenda (*A*)

```
{ (Variable
                entity-instance: menu-1
                property: hasCourses
                property-value: {mainCourse-1
                                 (AntiPasti :min 0 :max 1)
                                 (Dessert :min 0 :max 1)})
... }
```



SO: A → V

Variable (*V*)

```
(Variable
                entity-instance: menu-1
                property: hasCourses
                property-value: {mainCourse-1
                                 (AntiPasti :min 0 :max 1)
```

Select one decision that shall be made next.

Heuristic Operator (HO)

Variable (*V*)

```
(Variable
   entity-instance: menu-1
   property: hasCourses
   property-value: {mainCourse-1
                    (AntiPasti :min 0 :max 1)
                    (Dessert :min 0 :max 1)})
```

Computed by an heuristic operator (*HO*), e.g.,“ask the user”!

HO: V → R_V

Reduced Variable (R_V)

Result of a decision

```
(Variable
   entity-instance: menu-1
   property: hasCourses
   property-value: {mainCourse-1
                    (AntiPasti :min 0 :max 1)
                    dessert-1})
```

Extends the number of instances, i.e., increases the configuration.
Allows for incremental requirement acquisition.

Entailment Operator (*EO*)

Computes a partial configuration which contains the reduced variable and all entailments of this reduction computed by some reasoning method. Here the constraint processing comes in!

Reduced Variable (R_V)

```
(Variable
  entity-instance: menu-1
  property: hasCourses
  property-value: {mainCourse-1
                   (AntiPasti :min 0 :max 1)
                    dessert-1})
```

Partial Configuration (PC_i)

```
{(Entity-Instance name: menu-1
                  entity-model: Menu
                  kindOfTaste: hearty
                  hasCourses: { mainCourse-1
                                (AntiPasti :min 0 :max 1)
                                (Dessert :min 0 :max 1)}) …}
```

EO: CM, PC_i, R_V → Pc_{i+1}

Partial Configuration (PC_{i+1})

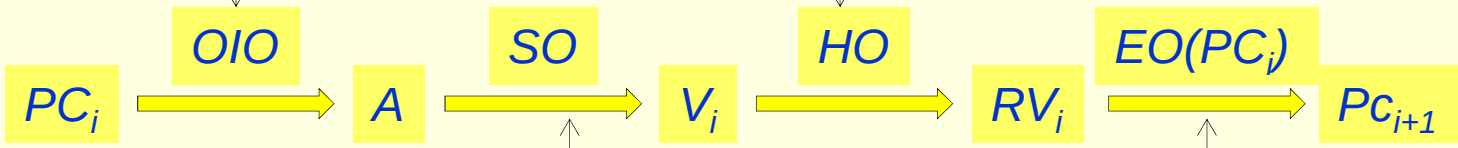
```
{(Entity-Instance name: menu-1
                  entity-model: Menu
                  kindOfTaste: hearty
                  hasCourses: { mainCourse-1
                                antiPasti-1
                                dessert-1}) …}
```

The Configuration Cycle

PC: Partial Configuration
OIO: Open Issue Operator
A: Agenda
SO: Select Operator
HO: Heuristic Operator
EO: Entailment Operator
V: Variable
RV: Reduced Variable

Comparing the partial configuration with the configuration model

E.g, incremental requirement acquisition



E.g, recommending a next decision

E.g, constraint processing

Loop: select decision, make decision, infer impacts

Discussion

- Under-specification of requirements is the usual case
- Configuration technologies should provide answers to following questions:
  - How is the model represented?
  - How are the requirements represented?
  - How is the configuration computed?

  And also questions, which make the difference to other AI-topics:
  - How are partial configurations represented?
  - How are decisions represented?
  - How are open issues computed?
  - How is/are the next decisions to be made selected?
  - How are conflicts handled?
  - How are retractions of decisions handled?

Discussion

- Don’t leave these questions to an outside user interface or architecture because of rules in the process:
  - Monotone decisions
  - Conflict resolution and decision retraction (moving in the sequence of decisions)
- Enhancements
  - Complex requirements
  - Select multiple decisions
  - Include simulation, optimization, manufacturing/realization in the loop
  - …

Thank you for your attention