Modeling, Representing, and Configuring Restricted Part-Whole Relations

Lothar Hotz [1]

Abstract. Part-whole relations are the backbone of configuration systems. In this paper, part-whole relations are combined with other relations and restrictions leading to here-called *restricted aggregates*. Depending on what is given, aggregates, parts, or/and relations between parts, different tasks have to be solved. General aggregation reasoning chunks are developed and represented with a configuration language. The approach is applied to the domain of constructing scene interpretations.

1 Introduction

Configuration is the task of composing parameterisable objects (*parts*) to wholes (*aggregates*) such that a given goal is fulfilled by the resulting construction. For this task, descriptions of aggregates and parts of a domain (*domain objects*) are given in a configuration model and a configuration tool is used to create a certain construction. An aggregate is typically described by its potential parts and further restrictions that have to be fulfilled between the parts. Thus, those *restricted* aggregates require that certain relations and predicates are true for their parts. Domain objects have parameters (i.e. relations to primitive data types) and n-ary relations to other domain objects, which can be given by a *task specification* or can be computed by the configuration tool (parameters are set and relations are established).

Configuration technologies are applied in technical areas like automotive, telecommunication, but also in software, services and construction of scene interpretations.

In the following, we develop general aggregation reasoning chunks that allow to construct aggregates from given parts and integrate parts in given aggregates. Furthermore, restricted aggregates, as they are considered here, require certain relations and predicates to be true for their parts. Also, when given such relations between parts, appropriate aggregates should be created that can contain those restricted parts. Thus, for diverse, here called, *key situations* (parts given with or without relations, aggregates given with or without parts) the general aggregation reasoning chunks will construct complete aggregates that guarantee the validity of the restriction.

As an example, we map the general aggregation reasoning chunks to a structure-based configuration language, such that the aggregate can be computed by an appropriate configuration tool [10]. Other representations e.g. with rule languages or usual programming languages are conceivable [11].

The general aggregation reasoning chunks are applied in the domain of Scene Interpretation, which has already been shown to be a configuration task [9]. In this domain, aggregates like *entrance* or *balcony* consist of certain parts like *door*, *railing*, *stairs*, *sign*, *window*. Parts have to fulfill restrictions like spatial relations of the kind *stairs below door*, *sign left-neighbor door* etc. An aggregate may additionally occur as part of a comprising aggregate like *facade*. The task is to construct a description with parts and aggregates of a scene given an image.[2]

In other approaches, compositional relations between parts and aggregates are considered from the epistemological point of view [19, 21, 15]. Beside technical aspects of how to represent parts and wholes with a given configuration language, the relationship between the compositional relation and further restrictions are less considered in those contributions. In this paper, the interplay between those relations are analyzed.

In principle, configuration technologies as described in [2, 4, 13, 17, 7, 12] provide a basic framework for constructing restricted aggregates. While this includes means for modeling, representing, and processing part-whole relations, it is seldom clarified how these facilities are *applied* or *used* for creating part-whole relations, or especially restricted part-whole relations.

This paper tries to further close this gap. Thus, we provide an analysis of possible aggregation situations and their representations by means of concepts and constraints. For this task, we first prescribe the *structure-oriented configuration approach*, which is applied here (Section 2), than consider the problem in more detail (Section 3). Section 4 describes the general solution of our approach for creating restricted part-whole relations and an implementation with a configuration language. Sections 5 and 6 describe some experiments and give a summary of the approach, respectively.

2 Structure-oriented Configuration Approach

As background, we follow the structure-oriented configuration approach as it is described by [1, 2, 3, 4, 5, 8, 16, 17, 18, 22]. While this approach has several variants we focus in the following on four separate knowledge types which can be used for modeling a certain domain:

Concept Hierarchy Domain objects are described using a highly expressive description language providing *concepts*, a taxonomical hierarchy (based on the `is-a` relation), and structural relations like the compositional hierarchy based on the `has-parts` relation. Parameters specify domain-object attributes with value intervals, sets of values (enumerations), or constant values. *Instances* selected for a concrete construction are instantiations of the concepts and represent concrete domain objects. Parameters

[1] HITeC e.V., University of Hamburg, Germany, email: hotz@informatik.uni-hamburg.de

[2] Our task differs to [20] in the fact that we construct descriptions of scenes (including aggregates and parts), while [20] computes spatial arrangements of non-aggregated scene objects.

and structural relations of a concept are also referred to as *properties* of the concept. When instantiated, the properties of an instance are initialized by the values or value ranges specified in the concepts.

Constraints *Constraints* pertaining to properties of more than one object are administered by a constraint net. *Conceptual constraints* are formulated as part of the configuration model. Conceptual constraints consists of a condition and an action part. The condition specifies a *structural situation* of instantiated concepts. If this structural situation is fulfilled by some instances, *constraint actions* that are formulated in the action part are instantiated to *constraints*. Constraint actions can represent restrictions between properties (i.e. *constraint relations*) or operations like `create-instance` (i.e. *constraint operations*). Constraints can be multi-directional, i.e. propagated regardless of the order in which constraint variables are instantiated or changed. At any given time, the remaining possible values of a constraint variable are given as intervals, value sets, or constant values.

Task Description A configuration task is specified in terms of an aggregate which must be configured (the goal) and possibly additional restrictions such as choices of parts, prescribed properties, etc. Typically, the goal is represented by the root node of the compositional hierarchy.

Procedural Knowledge The configuration process provides a stepwise composition of a construction. Each step is one of the following kinds of construction steps: *aggregate instantiation* (or *top-down structuring*), *part integration* (or *bottom-up structuring*), *instance specialization*, and *parameterization*. A step reduces a property value of an instance to a subset (*reduced value*) or finally to a constant (*fixed value*).

After each step the constraint net is optionally propagated. Configuration strategies are used to organize this configuration process in a declarative manner. For example, it is possible to prescribe phases of bottom-up or top-down processing conditioned on certain features of the evolving construction.

For every of the above mentioned knowledge types, a specific language is given which allows to express domain objects with their attributes and restrictions as well as the configuration process (see e.g. the Configuration Knowledge Modeling Language CKML described in [10]). In Figure 1, we sketch some constraint actions of the constraint language part of CKML that are needed for the following examples. The constraint operation `create-instance` and `ensure-relation` are the main mechanism for creating instances and relations between them. `create-instance` instantiates a concept after selecting one of given concept types.[3] Before establishing a relation between given instances, `ensure-relation` checks whether the relation already exists. Numeric constraint relations are used for comparing and computing parameter values given by constants, intervals, or enumerations in the typical mathematical form (which includes interval arithmetic).

For representing aggregates with their parts, spatial relations and restrictions, i.e. for representing *restricted aggregates*, concepts with their taxonomical and compositional relations as well as constraints can be applied, leading to *restricted aggregate models*. However, using this configuration technology for scene interpretation, the question arises, how these facilities can effectively be used both for modelling a domain (i.e. the analysis and in-depth understanding of the domain) and operationalizing this knowledge for the scene interpretation process.

Configuration systems considered here typically do not reason about concepts, but only about instances (for other approaches see e.g. [14]). Thus, the language facilities provided by such systems use instances for reasoning, e.g. conceptual constraints are fulfilled if an instance relation structure matches the structural situations. Thus, instead of description logics, which may also reason about concepts, in configuration systems, appropriate instances have to be created and for the created instances knowledge entities have to be inserted in the configuration model.[4] The question now is, what knowledge entities have to be inserted for reasoning about restricted aggregates?

create-instance	Creates a new instance for one of given concept types.
ensure-relation	Establishes a relation of of a given name between two instances.
less, greater, equal ...	Numeric constraint relations.

Figure 1. Predefined operations and relations (i.e. constraint actions) that are used in the following.

3 Requirements and Application Example

In the following, we consider aggregates locally described by their parts and restrictions. With a *local representation* of aggregates, all information about their parts are kept at one place, i.e. are not distributed over the configuration model. Especially restrictions between parts of different aggregates are not allowed. However, a part may be part of different alternative aggregates, e.g. a door might be part of an entrance or a balcony. Examples of locally defined aggregates are illustrated in Figure 2. It shows a general `Scene-Aggregate` and a specific `Entrance` aggregate with parts and several spatial relations between them. For the aggregate and parts, concept types (e.g. `Stairs`, `Door`), parameters (e.g. `size-x`), and relations between the parts (e.g. `belowNeighbor`, `overlap`) can be described. In the example, we consider spatial relations of parts, however, arbitrary n-ary restrictions between properties of the aggregate's parts can be specified with aggregate restrictions.

Beside the typical signature for concept types, parameters and relations, discriminators are given which describe a *sufficient condition* for the aggregate - if the discriminator holds, then the aggregate should exist. A conjunctive combination of parts and relations can be given as a discriminator. Thus, a discriminator can be only one part or several parts, as well as one or more relation between parts. In the example in Figure 2, six discriminators are given each consisting of one relation (e.g. `?b-s-c` representing `?stairs0 below ?canopy4`).

Each discriminator is a "unique selling point" of an aggregate in the sense that it distinguishes this aggregate from other aggregates. However, once an aggregate has been created, also other restrictions are examined, e.g. further required parts. These restrictions are *necessary conditions*. With this mechanism, complete descriptions of aggregates are created. Note that this may lead to hypotheses of parts

[3] Concept types can be selected e.g. by considering probabilities. In our current application domain, statistics can be computed from annotated images. However, this aspect is not further discussed in this paper.

[4] However, reasoning about concepts is restricted (e.g. n-ary roles are not provided in description logic systems) and thus, less used for configuration tasks.

which may exist or have to exist, if the aggregate exists. For example in the facade domain, a *door* of an *entrance* is hypothesized, if the above mentioned condition `?stairs0 below ?canopy4` is fulfilled through observed *stairs* and *canopy*.[5]

This reasoning is embedded in a backtracking environment as typically provided by configuration systems. Backtracking occurs when inferred decisions lead to a conflict, e.g. signified by the constraint net. In this case, conflict resolution mechanisms should be applied [10].

In the following, we assume that such aggregate models are manually created or learned from a set of given cases e.g. by Version Space Learning, see [6]. Especially discriminators (i.e. sufficient conditions) have to be identified by those methods.

Requirements for the general aggregation reasoning chunks. The aggregation chunks which we target should cover diverse *key situations*, which are given through the task specification or may come up through reasoning during the configuration process. The key situations are described in terms of given instances and properties. A property is *given*, if its value is fixed or reduced (see Section 2), otherwise it is *original* as it was specified in the concept (e.g. a parameter `pos-x-1` is `[0 inf]`). The chunks should:

- allow the construction of a new aggregate when one or more parts with relations or parameters are given (*bottom-up structuring*),
- allow the construction of new parts when an aggregate is given (*top-down structuring*),
- integrate given parts in given aggregates (*bottom-up integration*),
- use given parts for decomposing given aggregates (*top-down decomposition*),
- check given restrictions when all or some parts and aggregates are present (*aggregate consistency*),
- establish the described restrictions when parts and aggregates do not yet fulfill the restrictions (*restriction establishment*),
- determine appropriate types of aggregates and parts when certain relations but no specific type information are given (*object specialization*),
- select one aggregate types and parts when several alternatives are possible (*type selection*).

Modeling of restrictions. For modeling restrictions, we consider a two step approach: First, *physical parameters* are obtained through external systems, e.g. image processing systems, that supply geometric parameters like position and size of objects. These parameters cover *implicit relations* between objects, in particular *spatial predicates* as defined by predicates such as `above-p`. Those relations can be made explicit by establishing appropriate *explicit relations* between objects, such as `above`, `left`, `right` etc. Explicit relations are mainly used for describing restrictions on a high level as illustrated in Figure 2. They abstract from concrete numbers representing physical parameters.

Depending on the key situation, the relations are computed, e.g.:

- If an explicit relation between objects is given, the physical parameters should be changed, so that the geometry holds between the parameters. For example, in a two dimensional x/y coordination system, if an object $o1$ is identified to be below $o2$, the position parameter y of $o1$ should be higher than y of $o2$ (expecting the origin to be at the top left corner).
- If an implicit relation between physical parameters is given, the explicit relation between objects should be established. For example, in a two dimensional x/y coordination system, if the position parameters of $o1$ (i.e. y) is higher than y of $o2$, the relation $o1\ is\ below\ o2$ should be established.
- If the physical parameters are changed, the explicit spatial relations should be checked, if they hold.

[5] Discriminators should not be confused with mandatory or optional parts of an aggregate. In the example, all parts (*door*, *stairs*, and *canopy*) are mandatory for an *entrance*. For the existence of an entrance not all of them have to be observed, but only those mentioned in the discriminators. However, all mandatory parts will be created as hypotheses once the aggregate is instantiated.

```
(define-aggregate :name Scene-Aggregate
  :parameters
  ((size-x [0 inf])
   (size-Y [0 inf])
   (pos-x-1 [0 inf]) (pos-y-1 [0 inf])
   (pos-x-2 [0 inf]) (pos-y-1 [0 inf])
   (parts-top-left-x-variability [0 inf])
   (parts-top-left-y-variability [0 inf])
   (parts-bottom-right-x-variability [0 inf])
   (parts-bottom-right-y-variability [0 inf]))
  :parts
  ((:name ?parts :type Scene-Aggregate
                 :number-restriction [0 inf]))
  :restrictions
   ((:name ?variability
     :constraint
       (check-variability ?a ?parts))
    (:name ?bounding-box
     :constraint
       (check-bounding-box ?a ?parts))))

(define-aggregate :name Entrance
  :super Scene-Aggregate
  :parameters
  ((size-x [184 295])
   (size-Y [299 420])
   (parts-top-left-x-variability [7 131])
   (parts-top-left-y-variability [1 284])
   (parts-bottom-right-x-variability [7 131])
   (parts-bottom-right-y-variability [1 284]))
  :parts
  ((:name ?stairs0  :type Stairs)
   (:name ?door1    :type Door)
   (:name ?sign2    :type Sign)
   (:name ?railing3 :type Railing)
   (:name ?canopy4  :type Canopy))
  :restrictions
  ((:name ?bn-s-d   :relation belowNeighbor
    :subject ?stairs0 :object ?door1)
   (:name ?b-s-c    :relation below
    :subject ?stairs0 :object ?canopy4)
   (:name ?o-s-r    :relation overlap
    :subject ?stairs0 :object ?railing3)
   (:name ?bn-s-s   :relation belowNeighbor
    :subject ?stairs0 :object ?sign2)
   (:name ?an-d-s   :relation aboveNeighbor
    :subject ?door1 :object ?stairs0)
   (:name ?bn-d-c   :relation belowNeighbor
    :subject ?door1 :object ?canopy4)
  :discriminators
   ((?bn-s-d) (?b-s-c) (?o-s-r)
    (?bn-s-s) (?an-d-s) (?bn-d-c))))
```

Figure 2. Local aggregate description. A general `Scene-Aggregate`, which specifies restrictions that hold for all aggregates and a specific `Entrance` aggregate that inherits the general restrictions and properties. Question marked symbols (e.g. `?an-d-s`) indicate variables, which can bind objects or relations.

In Figure 3 the physical parameters and relations are listed, which are used in the following domain of Scene Interpretation.

4 Analysis of Aggregation Processing and Representation with a Configuration Language

In this section, we identify all key situations that may occur during the processing of restricted aggregates. This is done by permuting possible variabilities given by restricted aggregate models. Furthermore, for each key situation we provide representations based on facilities given by a configuration language.

As described above, depending on the given information about aggregates and parts, certain activities should be performed by the configuration process. For an aggregation, a key situation can be characterized by the presence or absence of an aggregate A instance (e.g. a

Spatial relations reflect the appropriate geometric relation: `overlap`, `inside`, `left-of`, `right-of`, `above`, `below`, `top-left`, `right-left`, `bottom-left`, `right-left`, `left-neighbor`, `right-neighbor`, `above-neighbor`, `below-neighbor`, `check-variability`.
Geometric parameters for describing bounding box and size of an object: `pos-x-1`, `pos-y-1`, `pos-x-2`, `pos-y-2`, `size-x`, `size-y`.
Spatial predicates check the geometric parameters, whether the spatial relations hold: `overlap-p`, `inside-p`, etc.

Figure 3. Predefined spatial relations, geometric parameters, and spatial predicates on geometric parameters for the Scene Interpretation domain. The `x-neighbor` relations indicate direct neighbors in the mentioned x direction.

No.	compositional rel. for A and p_i	spatial rel. for p_i	geometric para. for p_i
1	original	given	original
2	original	original	given
3	given	original	original
4	given	given	original
5	given	original	given
6	original	given	given
7	given	given	given

Figure 4. Possible situations for instances of one aggregate A and for potential parts p_i. If not given as constant or reduced, the value is an unchanged interval or not yet computed relations of the concept (indicated by $original$).

`balcony`) and its parts p_i (e.g. a `door`, a `window`). Hence, the variability of a key situation can be described as follows:

Instances A and p_i might be or might not be given. If neither A nor p_i are given, A may be constructed from the always given goal object g. This means, that an instance of a certain concept is created where all properties are original. If only A is given, appropriate p_i have to be constructed analogously and vice versa.

Concept type of A may be of a taxonomical *leaf* concept type or a *specializable* concept type. A *leaf* concept cannot be specialized further, and thus, only the aggregate parts and restrictions have to be computed. If a *specializable* concept type is given (e.g. as a general type like `Facade-Object`), a more specific type has to be computed by considering given or possible parts of the aggregate including their restrictions. For example, an instance of `Facade-Object` may be specialized to `Balcony`, if it has an instance of `Door` as a part.

Concept type of p_i is analogous to the above considerations.

Compositional relation between A **and** p_i might be or might not be given. If such relations are given, the restrictions given by A have to be checked for p_i. If the compositional relations are not given, they have to be established, if the p_i fulfill the restrictions given in the model of A. If A has further potential parts than p_i, those have to be created (i.e. part *hypotheses* are created, see [10]) and have to fit the given p_i.

Spatial relations between p_i might be or might not be given. If they are given, the geometric parameters have to have values according to the spatial relations, an appropriate aggregate has to be created (i.e. an aggregate *hypothesis*), and the compositional relations of that aggregate to the p_i have to be established. If spatial relations are not given, they must be computed from the spatial predicates and geometric parameters.

Geometric parameters of p_i and A might be or might not be given. If given, the appropriate spatial relations can be computed. If they are not given, geometric parameters can be computed from spatial relations.

Expecting that the instances with appropriate concept types are given, Figure 4 shows all other situations. Those are discussed in the following.

```
(define-conceptual-constraint
  :name Spatial-relation-from-predicate
  :structural-situation
  ((:name ?o1 :type Scene-Object)
   (:name ?o2 :type Scene-Object
              :relations
                ((self
                   #'(aboveNeighbor-p *it* ?o1)))))
  :action-part
  ((ensure-relation
      (above-neighbor ?o2 ?o1)
      (below-neighbor ?o1 ?o2))))
```

Figure 5. Generic conceptual constraint for Case 2 and Case 5 (*Case-Gen-1*). By checking spatial predicates the explicit spatial relations are established. `*it*` refers to the value of the relation, in the case of `self` to the object bound to `?o2`.

Each case has distinct impacts on the configuration process, i.e. distinct activities have to be performed. In general, in each case the missing information (in Figure 4 indicated by $original$) has to be computed from the fixed or reduced ones (indicated by $given$). Besides these activities, also the mappings to the representation facilities of the configuration language have to be specified. In the following, for each key situation representations are given as reasoning chunks.

```
(define-conceptual-constraint
  :name Spatial-relation-from-relation
  :structural-situation
  ((:name ?o1 :type Scene-Object)
   (:name ?o2 :type Scene-Object
              :relations
                ((above-neighbor *it* ?o1))))
  :action-part
  ((less (y-pos-2 ?o1) (y-pos-1 ?o2))
   (less (x-pos-1 ?o2) (x-pos-1 ?o1))
   (greater (x-pos-2 ?o2) (x-pos-2 ?o1))))
```

Figure 6. Generic conceptual constraint for Case 1 and Case 4 *Case-Gen-2*. By checking spatial relations the geometric parameters are computed by numeric constraints.

Case Reduction: From Case 2 to Case 6, and from Case 5 to Case 7. The mapping of given geometric parameters of p_i to spatial relations can be done with one generic conceptual constraint (*CC-Gen-1*) that holds for all types of parts (see Figure 5). There, arbitrary `Scene-Objects` (the super concept of every part or aggregate) are checked with spatial predicates in the structural situation and the appropriate spatial relations are established by `ensure-relation` in the action part. Because the spatial predicates can handle fixed and reduced values, this mapping is straight forward. With this conceptual constraint, also Case 2 and Case 5 can be processed as Case 6 and Case 7, respectively.

Case Reduction: From Case 1 to Case 6, and from Case 4 to Case 7. Similarly Case 1 and Case 4 can be reduced by introducing one generic conceptual constraint for mapping spatial relations to geometric parameters (see Figure 6, *CC-Gen-2*). The condition matches all p_i that are in the modeled spatial relation. The action part uses numeric constraints to compute the geometric parameters. However, in this case, because the geometric parameters are original, the mapping only reduces their intervals according to the spatial relations. This is done internally by constraint actions because of the underlying mathematics (see Section 2).

Case Reduction: From Case 6 to Case 7. The *compositional relation* of the parts to an aggregate instance have to be established. Each aggregate concept has to be checked that might be able to have parts with the given spatial relations. This can be achieved by using a conceptual constraint as shown in Figure 7. The structural situation of such a conceptual constraint describes the spatial relation which holds between the parts. The action part uses the `create-instance` constraint operation, which selects an aggregate concept out of a set of concepts. This set represents all aggregates that may have parts with the given spatial relations. According to the given concepts, one aggregate type A_s is selected (i.e. by considering probabilities). A new instance of A_s is created and the compositional relations between this new instance and the p_i are established. For every discriminator of an aggregate (in Figure 2 every spatial relation), one conceptual constraint of this kind is modeled (*Case-6-ccs*). Thus, this conceptual constraint can handle all situations where some parts are given which are not yet part of an aggregate.

```
(define-conceptual-constraint
  :name Entrance-creation
  :structural-situation
  ((:name ?stairs0 :type Stairs
          :relations
            ((part-of #'(free-p *it*))))
   (:name ?door1 :type Door
          :relations
            ((part-of #'(free-p *it*))
             (above-neighbor ?stairs0))))
  :action-part
  ((create-instance (Entrance Terrace)
                    (part-of ?stairs0)
                    (part-of ?door1))))
```

Figure 7. Conceptual constraints for creating a compositional relation with a new aggregate (i.e. an example for a *Case-6-ccs*). The conceptual constraint matches all stairs and doors that are appropriately related and are not yet part of an aggregate (indicated by $free - p$). `create-instance` selects one appropriate aggregate (e.g. the most probable one), creates one instance of that aggregate (e.g. `Entrance`), and establishes the compositional relations.

Case 3: Only compositional relations between A and p_i are given. The spatial relations between p_i have to be computed from the given compositional relations. Here the conceptual constraints (*Case-3-ccs*) have the following form: The structural situation checks the compositional relation between the A and p_i. The action part establishes the spatial relations between the p_i. For each spatial relation of an aggregate one conceptual constraint of the form illustrated in Figure 8 is created.

Case 7: Compositional relations between A and p_i, spatial relations p_j and geometric parameters of p_j are given. All conceptual constraints match and check the given relations. If the compositional relations are fixed for the same parts as the spatial relations (i.e. if $p_i = p_j$) the *Case-3-ccs* can be used for ensuring the spatial relations - `ensure-relation` than only checks the spatial relations between the parts. *CC-Gen-1* computes the geometric parameters for the p_j.

If there exists a p_j which is not yet part of A (i.e. its `free-p`), but holds the spatial relations of A, the compositional relation can be established with a further type of conceptual constraint (*Case-4-ccs*, see Figure 9). Furthermore, there may be combinations where one part is already part of A and another is not, e.g. in Figure 9 `Stairs` may be part of Entrance while `Door` is not, or vice versa. For this reason, the predicate `free-or-in-agg-p` is introduced that checks, whether an object is part of no aggregate or of the indicated one (e.g. part of `Entrance`).

Thus, five different types of reasoning chunks are finally identified:

```
(define-conceptual-constraint
  :name Entrance-Spatial-relation
  :structural-situation
  ((:name ?e :type Entrance)
   (:name ?stairs0 :type Stairs
          :relations ((part-of ?e)))
   (:name ?door1 :type Door
          :relations ((part-of ?e))))
  :action-part
  ((ensure-relation
      (above-neighbor ?door1 ?stairs0)
      (below-neighbor ?stairs0 ?door1))))
```

Figure 8. Conceptual constraint for Case 3 (*Case-3-ccs*).

```
(define-conceptual-constraint
  :name Entrance-Spatial-relation
  :structural-situation
  ((:name ?stairs0 :type Stairs)
   (:name ?door1 :type Door
          :relations
            ((aboveNeighbor ?stairs0)))
   (:name ?e :type Entrance
          :relations
            ((has-parts
               #'(free-or-in-agg-p ?stairs ?doors1 *it*)
               #'(check-variability *it*
                                    ?stairs0 ?door1)
               #'(check-bounding-box *it*
                                    ?stairs0 ?door1)))))
  :action-part
  ((ensure-relation
      (part-of ?stairs0 ?e)
      (has-parts ?e ?stairs0))
   (ensure-relation
      (part-of ?door1 ?e)
      (has-parts ?e ?door1))))
```

Figure 9. Conceptual constraint for Case 4 (*Case-4-ccs*). The predicate `free-or-in-agg-p` checks, whether the objects are parts of no aggregate or already part of the aggregate. `check-variability` and `check-bounding-box` are further aggregate restrictions that have to hold.

1. Mapping between numeric parameters and explicit relations (quantitative/qualitative mapping). This mapping is done by the conceptual constraint *Case-Gen-1*.
2. Mapping between explicit relations and numeric parameters (qualitative/quantitative mapping). This mapping is done by the conceptual constraint *Case-Gen-2*.
3. Creating new aggregates from given discriminative explicit relations (discriminators). This mapping is done by one conceptual constraint for each discriminator of an aggregate (*Case-6-ccs*).
4. Checking if restrictions between parts hold for parts that are elements of an aggregate. This mapping is done by one conceptual constraint for each discriminator of an aggregate (*Case-3-ccs*).
5. Integrating appropriate parts in existing aggregates while considering aggregate restrictions. This mapping is done by one conceptual constraint for each discriminator of an aggregate (*Case-4-ccs*).

5 Experiments in the Scene Interpretation Domain

We tested the previously described representation for the construction of descriptions of facade scenes. In Figure 10, left, primitive parts like *windows*, *stairs* are shown. These parts are aggregated to *balconies* and an *entrance* (see Figure 10, right). The experiments further showed that the selection of the domain-dependent predicates like `check-variability` and appropriate discriminator (see Figure 2) are very important. If several aggregates of the same type have to be considered (see Figure 11, right), the identification of the corresponding primitive parts are computed by those predicates.

6 Discussion and Summary

The generic aggregation reasoning chunks presented in this paper have the following properties:

- They distinguish between numeric, quantitative parameters typically given in databases or sensors, and qualitative relations which are used in abstract aggregation models.
- They compute all kinds of entities, i.e. aggregates, parts, relations and parameters depending on the given information.
- They are generic, i.e. they do not depend on the domain used in the examples, but can be applied to any domain with restricted aggregates, e.g. also to domains with temporal relations.

Because of the expressive language used in configuration technologies, domain restrictions with numeric, interval-based constraints and n-ary constraints can be used. Also, the configuration language with concepts and constraints used in our work can be mapped to similar representation facilities like classes, rules, and functions of other configuration approaches. Thus, the paper provides general modeling and representation facilities used for composing restricted aggregates.

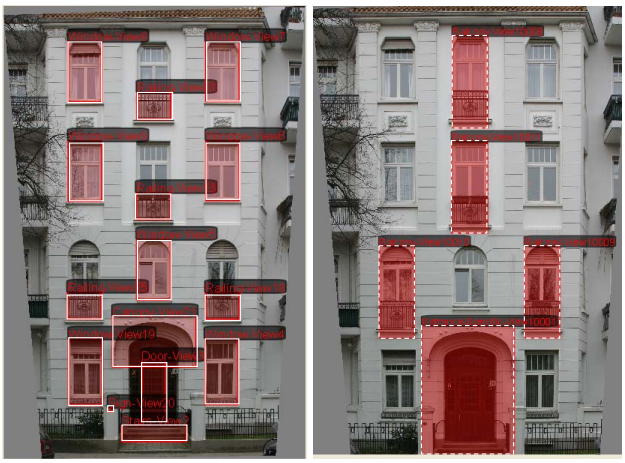

Figure 10. Left: Primitive facade objects here provided by annotation. Right: Constructed aggregates of type `entrance` and `balcony`. The annotated primitives and the automatically created aggregates are highlighted for presentation reasons.

Figure 11. Further example with primitives and several aggregates of one type.

ACKNOWLEDGEMENTS

This research has been supported by the European Community under the grant IST 027113, eTRIMS - eTraining for Interpreting Images of Man-Made Scenes.

REFERENCES

[1] R. Cunis, A. Günter, I. Syska, H. Peters, and H. Bode, 'PLAKON - an Approach to Domain-independent Construction', in *Proc. of Second Int. Conf. on Industrial and Engineering Applications of AI and Expert Systems IEA/AIE-89*, pp. 866–874, (June 6-9 1989).

[2] A. Günter, *Wissensbasiertes Konfigurieren*, Infix, St. Augustin, 1995.

[3] A. Günter and L. Hotz, 'KONWERK - A Domain Independent Configuration Tool', *Configuration Papers from the AAAI Workshop*, 10–19, (July 19 1999).

[4] A. Günter and C. Kühn, 'Knowledge-based Configuration - Survey and Future Directions', in *XPS-99: Knowledge Based Systems, Proceedings 5th Biannual German Conference on Knowledge Based Systems*, ed., F. Puppe, Springer Lecture Notes in Artificial Intelligence 1570, Würzburg, (March 3-5 1999).

[5] A. Haag, 'Sales Configuration in Business Processes', *IEEE Intelligent Systems*, (July/August 1998).

[6] J. Hartz and B. Neumann, 'Learning a knowledge base of ontological concepts for high-level scene interpretation', in *International Conference on Machine Learning and Applications*, Cincinnati (Ohio, USA), (December 2007).

[7] M. Heinrich and E. Jüngst, 'A Resource-based Paradigm for the Configuring of Technical Systems from Modular Components', in *Proc. of 7th IEEE Conf. on Artificial Intelligence for Applications (CAIA'91)*, pp. 257–264, Miami Beach, Florida, USA, (February 24-28 1991).

[8] L. Hotz, 'Configuring from Observed Parts', in *Configuration Workshop, 2006*, eds., C. Sinz and A. Haag, Workshop Proceedings ECAI, Riva del Garda, (2006).

[9] L. Hotz and B. Neumann, 'SCENIC Interpretation as a Configuration Task', Technical Report B-262-05, Fachbereich Informatik, University of Hamburg, (March 2005).

[10] L. Hotz, K. Wolter, T. Krebs, S. Deelstra, M. Sinnema, J. Nijhuis, and J. MacGregor, *Configuration in Industrial Product Families - The ConIPF Methodology*, IOS Press, Berlin, 2006.

[11] M. Jing and H. Boley, 'Interpreting SWRL Rules in RDF Graphs', *Electronic Notes in Theoretical Computer Science*, **151**, 53–69, (2006).

[12] D. Margo and P. Torasso, 'Interactive Configuration Capability in a Sale Support System', in *Proc. of Configuration Workshop, 17th International Joint Conference on Artificial Intelligence (IJCAI'01)*, pp. 57–63, Seattle, USA, (August 2001).

[13] J. McDermott, 'R1: A Rule-based Configurer of Computer Systems', *Artificial Intelligence Journal*, **19**, 39–88, (1982).

[14] S. Mittal and F. Frayman, 'Towards a Generic Model of Configuration Tasks', in *Proc. of Eleventh Int. Joint Conf. on AI IJCAI-89*, pp. 1395–1401, Detroit, Michigan, USA, (1989).

[15] S. Pribbenow, 'What's a Part? - On Formalizing Part-Whole Relations', in *Foundations of Computer Science*, volume 1337 of *Springer Lecture Notes in Computer Science*, pp. 399–406, (1997).

[16] K.C. Ranze, T. Scholz, T. Wagner, A. Günter, O. Herzog, O. Hollmann, C. Schlieder, and V. Arlt, 'A Structure-based Configuration Tool: Drive Solution Designer DSD', *14. Conf. Innovative Applications of AI*, (2002).

[17] T. Soininen, J. Tiihonen, T. Männistö, and R. Sulonen, 'Towards a General Ontology of Configuration', *Artificial Intelligence for Engineering Design, Analysis and Manufacturing (1998), 12*, 357–372, (1998).

[18] M. Stumptner, 'An Overview of Knowledge-based Configuration', *AI Communications*, **10(2)**, 111–126, (1997).

[19] A. Varzi, 'Parts, Wholes, and Part-Whole Relations: The Prospects of Mereotopology', *Data and Knowledge Engineering*, **20(3)**, 259–286, (1996).

[20] D.L. Waltz, 'Understanding Scenes with Shadwos', in *The psychology of computer vision*, pp. 19–91, New York, (1975). McGraw-Hill.

[21] P.H. Winston, R. Chaffin, and D. Herrmann, 'A Taxonomy of Part-Whole Relations', *Cognitive Science*, **11**, 417–444, (1987).

[22] B. Yu and H.J. Skovgaard, 'A Configuration Tool to Increase Product Competitiveness', *IEEE Intelligent Systems*, **13**(4), 34–41, (July 1998).