Proposal for an extended specification of the COINVENT system

IIIA-CSIC

1 Introduction

This document proposes an extension of the initial specification of the COINVENT system described in D8.1 [1] by introducing a new component **amalgam**, that is an *amalgamation*-based blending module, and its API.

The **amalgam** component searches, selects and blends possible generalisations of input spaces in order to return a preordered set of blends. The module depends on other components of the COINVENT system already specified in D8.1:

- the **base** component for computing the *generic spaces*;
- the **blend** component for computing a *blend*;
- the **consistency** component for checking that a blend is consistent (ideally for checking that a blend satisfies certain properties);
- the **generalisation**[1] component to find a set of elementary generalisations for an input space.

Additionally, we propose a terminology to be used in the COINVENT project.

2 Terminology

We propose to use the following terminology and to update the terms used in D8.1 [1] in the following way:

- "base" component → "genspace" component
- "weaken" component → "generalisation" component
- "input concept" → "input space"
- "base concept" → "generic space"
- "blended concept" → "blend"
- "weaker" / "weakening" → "generalised" / "generalisation"

[1] In D8.1, this component is called **weaken**.

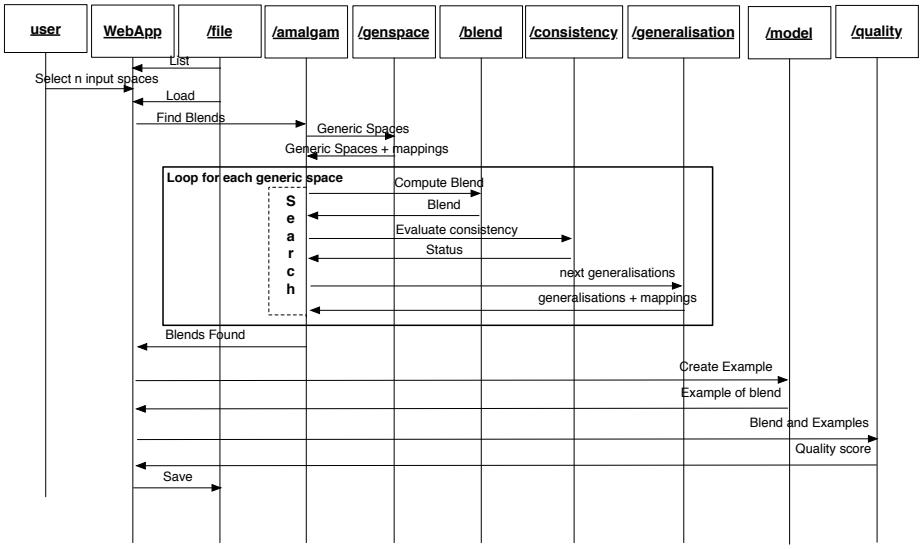

Figure 1: UML sequence diagram of the extended specification of the COINVENT system

3 COINVENT workflow

Our proposal to extend a typical COINVENT session described in D8.1 [1, Figure 1] is shown by the UML sequence diagram in Figure 1.

The user browses and selects several input spaces from a rich background of domain concepts. A set of blends is then developed by amalgamating the input spaces. Examples of the blends are generated. Finally, the blends' quality is evaluated based on the examples.

4 Core Object: The Blend Diagram in Progress

We adopt and extend the Blend Diagram and Blend Diagram in Progress described in D8.1 [1, Section 3] as follows:

- We allow more than two input spaces to have the same generic space.
- We allow more than one generic space.

A "Blend Diagram" consists of:

1. A Blend
2. A set of Input Spaces
3. Generalisations of the Input Spaces, e.g. a mapping from a Generalised Input Space to the Input Space by using elementary generalisation refinement operators (see Section 5.2).
4. Generic Spaces. A generic space is a common space for some of the Input Spaces (at least two). If there are generalisations, then a Generic Space is a common space for the generalised input spaces (not the originals).

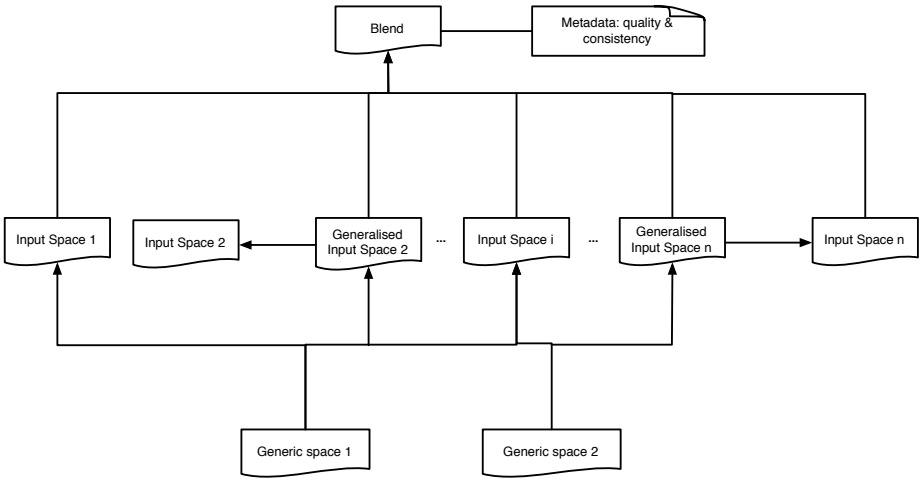

Figure 2: Blend diagram of the COINVENT system

5. Mappings from the Generic Spaces to each of the (Generalised) Input Spaces.
6. Mappings from each of the (Generalised) Input Spaces to the Blend.
7. Optional metadata about the Blend, such as evaluation scores, proof obligations, or relationship to other Blends.

A "Blend Diagram in Progress" is a Blend Diagram where some parts are missing.

5 Components

The extended specification of the COINVENT system will be made up of the components described in D8.1, but with the following changes:

- there is an **amalgam** component which looks for blends by orchestrating other COINVENT components such as the **genspace**, **blend**, **consistency** and **generalisation** component.
- the **weaken** component is replaced by a **generalisation** component which makes use of different language-dependent modules that implement several elementary generalisation refinement operators.
- the **base** component is renamed to **genspace** component according to the terminology proposed.

5.1 Amalgam: given a Blend Diagram in Progress it computes blends

The **amalgam** component can be thought as the core blending module of the COINVENT system. It uses the *amalgam* approach of blending [2], where a set of input spaces to blend is given, and each input space is generalised just as

much as required such that the blend of the generalised input spaces satisfies some computable criteria (e.g. consistency and/or quality criteria).

This component implements an interleaved evaluation-search process based on Answer Set Programming (ASP) [3, 4]. The search selects and amalgamates generalisations of input spaces, and returns a preordered set of blends. The search can be bounded by generic spaces that can be obtained by anti-unification, calling the **genspace** API (the HDTP software) or by other means. The blends, on the other hand, are computed as *colimits* by using the **blend** component API (HETS). For checking the consistency of a blend, the **consistency** component is used.

When a blend does not satisfy certain some criteria, **amalgam** needs to know how to generalise the input spaces used in the blending. The generalisation of an input space is achieved by the **generalisation** component which returns a set of possible generalised input spaces (associated with a priority level).

5.2 Generalisation: Given an inconsistent Blend Diagram, generalised the input spaces

The **generalisation** component finds the elementary generalisations for an input space and the corresponding monomorphisms (mappings).

Whilst this component is a common entry for generalisation requests, the generalisations and the mappings are generated by language-dependent modules that implement several *elementary generalisation refinement operators*.[2]

Elementary generalisation refinement operators may be characterised according to some properties, for instance [5, 6]:

- *completeness*: there are no generalisations of an input space which are not generated by the operator;
- *properness*: an input space is not equivalent to any of its generalisations;
- *local finiteness*: the number of generalisations generated for any input space by the operator is finite.

Depending on the language and semantics of the generalisation operators, these properties can be all met or not. This can have some impact on the implementation. The definition of the generalisation operators for CASL and OWL is an important topic under research.

Some elementary generalisation operators for a CASL generalisation module we identified so far are:

- **rmAx**(I, ax): if the axiom ax exists in the input space I, then it removes ax from I;
- **rmOp**(I, op): if the operator op exists in the input space I and all axioms used by op have already been removed, then it removes op from I.

The set of elementary generalisation refinement operators above is complete, finite and proper.

[2]The reason of having more than one generalisation module is that the languages chosen for knowledge representation in COINVENT, such as CASL and OWL, have different ways for generalising an input space. For each language, the generalisation of an input space relies on specific elementary generalisation refinement operators. Therefore, implementations of such operators must be provided for each language.

A Components API

The components API for the extended specification of the COINVENT system will be made up of the components API described in D8.1, but with the following changes or additions:

- the /blend API should allow as input an array of (generalised) input spaces and an array of mappings from the generic spaces to the input spaces. Consequently, in the output, it should allow a set of mappings from the inputs to the blends (see Annex A.2).
- the /genspace API should allow as input an array of input spaces, an optional array of generics spaces and of mappings from generic spaces to input spaces. As output, it should allow a set of mappings from the generic spaces to the input spaces (see Annex A.3).
- the /amalgam API takes as input an array of input spaces, an optional background knowledge and an optional natural number to limit the search. It returns a set of blends and the mappings from the inputs to the blends (see Annex A.4).
- the /generalisation API takes as input an input space and returns a set of generalised input spaces with their mappings to the input spaces. If a generic space is provided as input, then it also returns the mapping from the generic space to the generalised input spaces (see Annex A.5). It is worthy to clarify here that this API is a common API for the generalisation requests, but that more specific APIs, one for each language-dependent module implementing elementary generalisation refinement operators, are needed (see Annex A.6).

A.1 Common/Convention

We adopt the same conventions of D8.1.

A.2 /blend API

```
Default implementation: HETS
Default end point: http://coinvent.soda.sh:8400/blend/hets

Parameters:
- lang: owl | casl
- inputs: {concept[]}
- genspaces: {concept[]}
- genspaces_inputs: {mapping[]} from generic spaces to the inputs

Response-cargo:
{
blend: {concept} is a blend of the inputs
inputs_blend: {mapping[]} from inputs to blend
}
```

A.3 /genspace API

```
Default implementation: HDTP
Default end point: http://coinvent.soda.sh:8400/base/hdtp
Parameters:
- lang: owl | casl
- inputs: {concept[]}
- genspaces: {?concept[]}
- genspaces_inputs: {?mapping[]} from generic spaces to inputs
- cursor: {?url} For requesting follow-on results.

Response-cargo:
{
genspaces: {concept[]} which is an array of generic spaces
genspaces_inputs: {mapping[]} from generic spaces to inputs
}
```

A.4 /amalgam API

```
Default implementation: ASP
Default end point: http://coinvent.soda.sh:8400/amalgam/asp

Parameters:
- lang: owl | casl
- inputs: {concept[]}
- threshold: {?int}
- background_knowlegde: {concept}

Response-cargo:
{
blends: {concept[]} which is an array of blends
inputs_blend: {mapping[]} from inputs to the blends
}
```

A.5 /generalisation API

```
Default implementation: currently ASP
Default end point: http://coinvent.soda.sh:8400/generalisation/asp

Parameters:
- lang: owl | casl
- input: {concept}
- genspace: {?concept}
- genspace_input: {?mapping} from genspace to input

Response-cargo:
{
geninputs: {concept[]}
geninputs_input: {mapping[]} from the generalised inputs to the input space
genspace_geninputs: {?mapping[]} from the generic space to the generalised input spaces
```

```
}
```

A.6 /nextGeneralisationsCASL API

```
Default implementation: currently ASP
Default end point: http://coinvent.soda.sh:8400/generalisation/casl

Parameters:
- input: {concept} in CASL
- genspace: {?concept} in CASL
- genspace_input: {?mapping} from genspace to input in CASL

Response-cargo:
{
geninputs: {concept[]}
geninputs_input: {mapping[]} from the generalised inputs to the input space
genspace_geninputs: {?mapping[]} from the generic space to the generalised input spaces
}
```

References

[1] Winterstein, D., Maclean, E., Codescu, M.: D8.1 initial specification of the system. Technical report, COINVENT project (2014)

[2] Ontañón, S., Plaza, E.: Amalgams: A formal approach for combining multiple case solutions. In Bichindaritz, I., Montani, S., eds.: Case-Based Reasoning. Research and Development, 18th International Conference on Case-Based Reasoning, ICCBR 2010, Alessandria, Italy, July 19-22, 2010. Proceedings. Volume 6176 of Lecture Notes in Computer Science., Springer (2010) 257–271

[3] Gebser, M., Kaminski, R., Kaufmann, B., Schaub, T.: Answer Set Solving in Practice. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan and Claypool Publishers (2012)

[4] Gebser, M., Kaminski, R., Kaufmann, B., Schaub, T.: Clingo = ASP + control: Preliminary report. CoRR **abs/1405.3694** (2014)

[5] Ontañón, S., Plaza, E.: Similarity measures over refinement graphs. Mach. Learn. **87**(1) (2012) 57–92

[6] van der Laag, P.R., Nienhuys-Cheng, S.H.: Existence and nonexistence of complete refinement operators. In Bergadano, F., De Raedt, L., eds.: Machine Learning: ECML-94. Volume 784 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (1994) 307–322