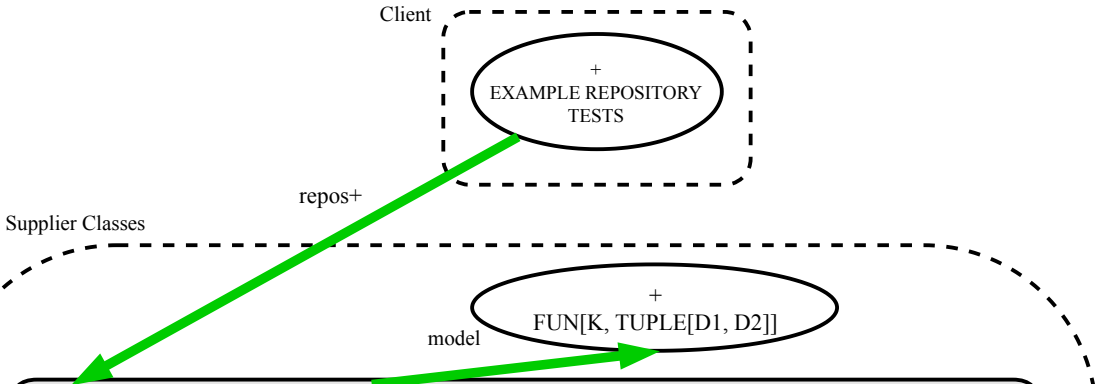

REPOSITORY[K, D1, D2]+

feature --Attributes
keys: LINKED_LIST[K]
data_items_1: ARRAY[D1]
data_items_2: HASH_TABLE[D2, K]

feature --Abstraction function
model: FUN[KEY, TUPLE[d1: DATA1; d2: DATA2]]

feature --Constructor
make
ensure
  empty_repository: model.is_empty

feature --Mutators
check_in (d1: D1; d2: D2; k: K)
require
    non_existing_key: not model.domain.has (k)
ensure
    repository_count_incremented: model.count = old model.count + 1
    data_set_added: model ~ (old model.deep_twin).extended (k, [d1, d2])
    others_unchanged: ∀i : i ∈ model: (i.first /~ k) implies ((old model.deep_twin).domain.has (i.first) and
        (old model.deep_twin).range.has ([i.second.d1, i.second.d2]))

check_out (k: KEY)
require
  existing_key: ∃i : i ∈ model.domain : i ~ k
ensure
   repository_count_decremented: model.count = old model.count - 1
   key_removed: model ~ (old model.deep_twin).domain_subtracted_by (k)
   others_unchanged: ∀j : j ∈ old model : (j.first /~ k) implies (model.domain.has (j.first)
            and model.range.has ([j.second.d1, j.second.d2]))

count
ensure
   correct_result: Result = model.count

matching_keys (d1: DATA1; d2: DATA2): ITERABLE[KEY]
ensure
  correct_keys_are_in_result: ∀j, ∀k: k ∈ result, j ∈ model:
            (j.item.first ~ k.item) implies (j.item.second.d1 ~ d1 and j.item.second.d2 ~ d2)
  result_contains_correct_keys_only: ∀j: j ∈ result: model.range_restricted_by ([d1, d2]).domain.has (j.item)

invariant
consistent_keys_data_items_counts:
keys.count = data_items_1.count ∧
keys.count = data_items_2.count