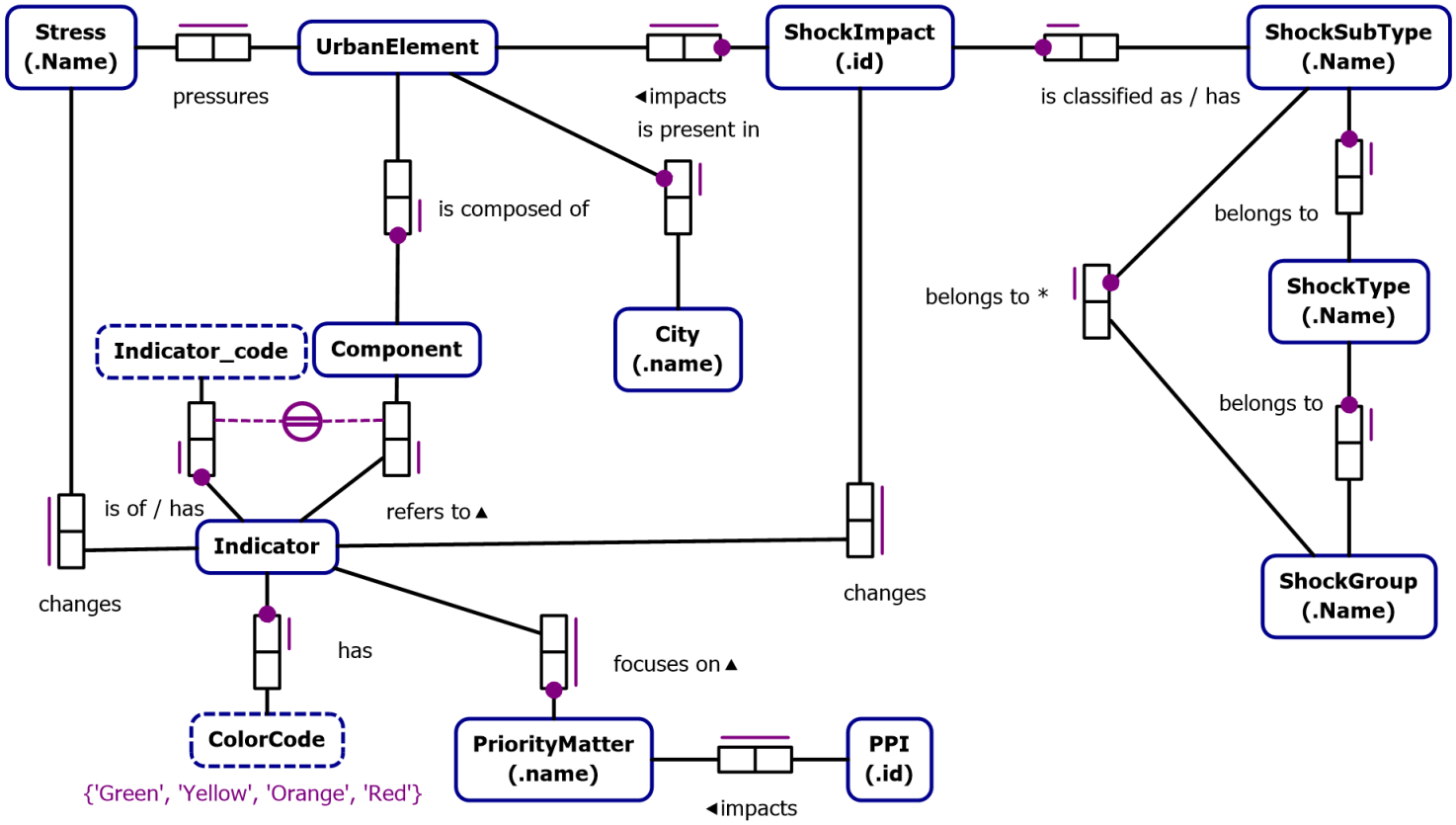
un-crpt-model_1

City Resilience Profiling Tool - Domain Ontology

We rely on the following definitions of concepts in the CRPT domain:

- **Urban Elements**: categorization of main aspects enabling life in the city, from Built Environment and Infrastructure to Municipal Public Services or Economy.
- **Component**: subcategories of urban elements
- **Indicators**: Indicators dedicated to assessing and characterizing each Urban Element performance (e.g. coverage and continuity of function of services).
- **Shocks**: uncertain, abrupt, or long-onset events that have potential to impact upon the purpose or objective of an urban system.
- **Stresses**: chronic and ongoing dynamic pressures originated within an urban system with potential for cumulative impacts on the ability and capacity of the system to achieve its objectives (e.g. environmental pollution, vegetation loss, informal economy).
- **Challenges**: long-term contextual changes and pressures originated outside of the urban system, also undermine the city's capacity for sustainability and resilience (e.g. climate change effects, intergovernmental coordination).
- **Priority Matters**: main areas of concern for the city being analyzed (economic performance, water cycle mismanagement).
- **Policies, plans and initiatives**: policies and plans in place or in development that may affect elements in the city (e.g: national, regional or local regulations).
- **Stakeholders**: local government, public and private sector entities, civil society.

To present the relationships between these concepts, we have created the following ontology in ORM:



crpt ontology implementation

install un-crpt-model-component

```
entity Component row_to_component = row_from[components_csv]
def component_row = transpose[row_to_component]
def component = cell_from[component_row, components_csv]

def component_city = component:city
def component_element_code = component:element_code
def component_code = component:component_code
def component_name = component:component_name
```

install un-crpt-model-indicator

```
entity Indicator row_to_indicator = row_from[priority_matters_indicators_csv]
def indicator_row = transpose[row_to_indicator]
def indicator = cell_from[indicator_row, priority_matters_indicators_csv]

entity Indicator row_to_indicator_2 = row_from[priority_matters_shocks_indicators_csv]
def indicator_row_2 = transpose[row_to_indicator_2]
def indicator = cell_from[indicator_row_2, priority_matters_shocks_indicators_csv]

def indicator_city = indicator:city
```

```
def indicator_prioritymatter = indicator:priority_matter . priority_matter_name_priorit
def indicator_shock_subtype = indicator:shock . from_shock_subtype_name
def indicator_urbanelement = indicator:element_code . (transpose[urban_element_code])
def indicator_component = indicator:component_code . (transpose[component_code])
def indicator_code = indicator:indicator_code
def indicator_description = indicator:indicator_description
def indicator_value = indicator:indicator_value
```

install | un-crpt-model-organization

```
entity Organization row_to_organization = row_from[stakeholders_description_csv]
def organization_row = transpose[row_to_organization]
def organization = cell_from[organization_row, stakeholders_description_csv]

def organization_city = organization:city
def organization_name = organization:organization
def organization_description = organization:description
```

install | un-crpt-model-ppi

```
entity PPI row_to_ppi = row_from[ppi_csv]
def ppi_row = transpose[row_to_ppi]
def ppi = cell_from[ppi_row, ppi_csv]

def ppi = ppi:type

def Policy = p: ppi(p, "Policy")
def Plan = p: ppi(p, "Plan")
def Initiative = t: ppi(t, "Initiative")

def ppi_city = ppi:city
def ppi_description = ppi:description
def ppi_year = ppi:year
def ppi_status= ppi:status
def ppi_category= ppi:category
def ppi_url= ppi:url
```

install | un-crpt-model-prioritymatter

```
entity PriorityMatter row_to_prioritymatter = row_from[priority_matters_csv]
def prioritymatter_row = transpose[row_to_prioritymatter]
def prioritymatter = cell_from[prioritymatter_row, priority_matters_csv]
```

```
def prioritymatter_city = prioritymatter:city
def prioritymatter_name = prioritymatter:priority_matter

def prioritymatter_shock_subtype(pm, sh) =
    shocks_prioritymatters_csv(:priority_matter, r, pm_n) and
    shocks_prioritymatters_csv(:shock, r, sh_n) and
    prioritymatter:priority_matter(pm, pm_n) and
    sh = from_shock_subtype_name[sh_n]
    from pm_n, sh_n, r
def shock_subtype_prioritymatter = transpose[prioritymatter_shock_subtype]
def priority_matter_name_prioritymatter = transpose[prioritymatter:priority_matter]
```

install | un-crpt-model-relationship_role

```
entity RelationshipRole from_role_name = urbanunderperformance_stakeholders_csv:role
```

install | un-crpt-model-shock_classifiers

```
entity SockGroup from_shock_group_name = {
    "BIOLOGICAL";
    "NATURAL";
    "ENVIRONMENTAL";
    "SOCIETAL";
    "TECHNOLOGICAL";
    "COMPLEX";
}

entity ShockType from_shock_type_name = {
    "INFECTIOUS DISEASES";
    "INFESTATION";
    "DROUGHT";
    "EXTREME METEOROLOGICAL CONDITIONS";
    "WILDFIRE";
    "EARTHQUAKE";
    "MASS MOVEMENT";
    "VOLCANIC ACTIVITY";
    "FLOOD";
    "STORM";
    "WAVE ACTION";
    "WATER-SOIL DEGRADATION";
    "AIR POLLUTION";
    "EROSION";
    "BIODIVERSITY LOSS";
    "SOCIO-ECONOMIC SHOCKS";
    "SOCIO-SPATIAL SHOCKS";
    "SOCIO-CULTURAL SHOCKS";
    "SOCIO-POLITICAL SHOCKS";
    "CRIME";
    "CYBER-ATTACK";
```

```
"TERRORISM";
"CONFLICT";
"INDUSTRIAL & MINING INCIDENT";
"NON-INDUSTRIAL INCIDENT";
"FAILURE OF INFRASTRUCTURE & SERVICES";
```

install un-crpt-model-shock_impact

```
entity ShockImpact row_to_shock_impact = row_from[shocks_csv]
def shock_impact_row = transpose[row_to_shock_impact]
def shock_impact = cell_from[shock_impact_row, shocks_csv]
// Similarly, here could be added shock occurrences from other locations as well
// by employing the pattern shown in lines 1–3

def shock_impact_id = shock_impact:id
def shock_subtype_to_impact = transpose[shock_impact_subtype]
def shock_impact_subtype(s, st) =
    shock_impact:subtype(s, stn) and
    st = from_shock_subtype_name[stn]
    from stn
def shock_impact(s, t) =
    shock_impact_subtype(s, st) and
    shock_subtype_belongs_to_shock_type(st, t)
    from st
def shock_impact_group(s, g) =
    shock_impact(s, t) and
    shock_type_belongs_to_shock_group(t, g)
    from t

def shock_impact_city = shock_impact:city
def shock_impact_date = shock_impact:date

```

```
def shock_impact_people_directly_affected = shock_impact:people_directly_affected
def shock_impact_people_indirectly_affected = shock_impact:people_indirectly_affected
def shock_impact_caused_shock = shock_impact:triggered_shock
def shock_impact_caused_by_shock = transpose[shock_impact_caused_shock]
def shock_impact_affected_urban_elements_list = shock_impact:urban_elements
def shock_impact_affected_urban_elements[s] = string_trim[split_string[shock_impact_aff
def shock_impact_number_of_casualties = shock_impact:casualties
def shock_impact_number_of_injured = shock_impact:injured
def shock_impact_number_of_displaced = shock_impact:displaced
def shock_impact_loss_of_working_days = shock_impact:loss_of_working_days
def shock_impact_loss_of_jobs = shock_impact:loss_of_jobs
```

install | un-crpt-model-stress

```
entity StressType row_to_stress = row_from[stresses_csv]
def stress_row = transpose[row_to_stress]
def stress = cell_from[stress_row, stresses_csv]

def stress_city = stress:city
def stress_name = stress:stress
def stress_from_name = transpose[stress_name]

// entity StressType row_to_stress2 = row_from[urbanunderperformance_stresses_indicator
// def stress_row2 = transpose[row_to_stress2]
// def stress = cell_from[stress_row2, urbanunderperformance_stresses_indicators_csv]

// city;priority_matter;stress;stressor;element_code;component_code;indicator_code;indi

// entity StressType stress_from_name = stressors_stresses_csv[:stress, _]
// entity StressType stress_from_name = stresses_prioritymatters_csv[:stress, _]
// entity StressType stress_from_name = stresses_urbanelements_csv[:stress, _]




def stress_urban_element_code(st, e_c) =
    urbanunderperformance_stresses_indicators_csv(:element_code, r, e_c) and
    urbanunderperformance_stresses_indicators_csv(:stress, r, st_n) and
    st = stress_from_name[st_n]
    from st_n, r

def stress_component_code(st, c_c) =
    urbanunderperformance_stresses_indicators_csv(:component_code, r, c_c) and
    urbanunderperformance_stresses_indicators_csv(:stress, r, st_n) and
    st = stress_from_name[st_n]
```

```
    from st_n, r

def stress_prioritymatter(st, pm) =
    urbanunderperformance_stresses_indicators_csv(:priority_matter, r, pm_n) and
    urbanunderperformance_stresses_indicators_csv(:stress, r, st_n) and
    prioritymatter:priority_matter(pm, pm_n) and
```

install un-crpt-model-stressor

```
entity StressorType stressor_from_name = stressors_stresses_csv[:stressor, _]

def stressor_stress(stor,stss) =
    stressor_from_name(stor_n, stor) and
    stress_from_name(stss_n, stss) and
    stressors_stresses_csv(:stressor, r, stor_n) and
    stressors_stresses_csv(:stress, r, stss_n)
    from stor_n, stss_n, r
def stress_stressor = transpose[stressor_stress]
```

install un-crpt-model-urbanelement

```
entity UrbanElementType row_to_urban_element = row_from[elements_csv]
def urban_element_row = transpose[row_to_urban_element]
def urban_element = cell_from[urban_element_row, elements_csv]

def urban_element_city = urban_element:city
def urban_element_code = urban_element:code
def urban_element_short_name = urban_element:short_name
def urban_element_name = urban_element:name
```

install un-crpt-model-lib

```
/**
  * Template for extracting an attribute of an entity from a CSV file when
  *   that file conforms to the entity master-data pattern.
  */
@inline
def row_from[C](row) = C(_, row, _)

@inline
def row_from[C](file_index, row) = C(file_index, _, row, _)
```

```
@inline
def row_from_by_key[KEYSPEC,C](key) = C(ks, _, key) and KEYSPEC(ks) from ks

@inline
def cell_from[R,C](attribute, p, v) =
    R(p, row) and
    C(attribute, row, v) and
    (String(v) and not empty_string(v)
        or
        Date(v) and not empty_date(v) and not default_end_date(v)
        or
        not String(v) and not Date(v))
    from row


@inline
def cell_from[R,C](attribute, p, v) =
    R(p, file_idx, row) and
    C(file_idx, row, attribute, v) and
    (String(v) and not empty_string(v)
        or
        Date(v) and not empty_date(v) and not default_end_date(v)
        or
        not String(v) and not Date(v))
    from row, file_idx

@inline
```