Pattern Extraction

# What Is required?

* An input is an incident model that is based on the incident pattern meta-model.
  + A model contains specific entities and relationships between them.

# Process to extract a pattern. It can be as follows

* First, do an abstraction round for the entities only. Define a set of entities SE (Specific Entity set), then do a function *Q(se) = ae*, where *se* belongs to SE and *ae* belongs to AE (Abstract Entity set). However, the abstract entity set (i.e. AE) is created from the Q function and can be refined/changed or different sets can be created that correspond to the Q function.
  + What is Q function? How should we define it?
  + We could introduce **abstraction levels** in the system meta-model. Thus, if an entity is abstracted to a level, then all entities in a condition (pre or post) will be abstracted to the same level. What is an abstraction level? An abstraction level can be defined by the inheritance and association relations.
  + How about properties of an entity? Indication of the abstraction level at which it can exist could be a solution.
* Second, do an abstraction of the conditions of concrete activities defined. How abstraction should be done? Define **rules**. Abstraction rules can be defined over the relationships (containment and connectivity) of BRS statements in conditions.

## Entity abstraction

The goal of this step is to find a *suitable* abstraction of the assets defined in an incident instance model.

For each asset in the incident instance model, abstract the asset to an entity that corresponds to a predefined level of abstraction in the system meta-model.

**System meta-model defines levels of abstraction**: initially define three levels of abstraction in the meta-model. abstraction level:

* Level 1: most abstract. For example, physical asset. Defines properties that can be at this level (e.g., name, connectivity, and containment)
* Level 2: less abstract (more concrete). For example, smart device. Defines what properties can be at this level (e.g., status).
* Level 3: least abstract (most concrete). For example, smart light. Again, defines what properties can be at this level (e.g., model number).

**Process for abstracting assets.**

*For* each asset:

1. Determine at what level the asset is (level 1, 2, or 3).
2. *If* asset is least abstract (most concrete), then abstract the asset to the previous level. For example, if an asset is at level 3, it is abstracted to level 2.
   * This includes creating new asset that is from the chosen abstraction level and define the properties that are defined in the original asset and that are part of the current chosen level
     + includes removing properties that belong to the previous level, unless the modeller specifies that a property should be kept during abstraction (could be used for learning). For example, remove the “model number” property if going from level 3 to level 2.
     + and giving new name to the new asset.
   * *The meta-model could specify preferred abstraction level (class)*
3. *Else If* an asset is not in the least abstract level (most concrete), then keep the asset as is, but change the name (retain mapping between new and old names).

**Output**: an incident instance model with abstracted entities.

Since the preconditions and postconditions are based on the connectivity and containment relationships between assets and that these relationships are defined at the highest abstraction levels, then we assume that replacing the original assets with the abstracted assets **will keep the meaning of the relationships**

## Conditions/Actions Abstraction

We need to define the set of rules on which actions (this means also the pre-/post-conditions) can be abstracted.

* How to abstract containment relationship?
  + Example of rule: if B contains A in one activity, and in the next activity, C contains A, and B & C are connected, then we can abstract the two actions to one action where C contains A. this rule corresponds to movement in the smart building (e.g., between rooms).
* How to abstract connectivity relationship?
  + The same first example given above for containment can be applied for connectivity.
* What’s the relation between the two? Can abstracting one affect the other?

# What is abstraction?

We require to give a definition of what constitutes a level of abstraction in our meta-model. Abstraction levels are defined over the system meta-model. A paper that discusses abstraction levels in meta-models and their definition is published by Monperrus et al. [1]. They use set theory to give a definition of an abstraction level. Then they show it using an example.

See literature on process modelling abstraction as cited in Lara et al. [2] paper. There is also literature on system transitions abstraction which could be applied to the system analysis (just an idea). They mention that abstraction operations such as in BPMN can be defined at the level of the meta-model then applied to the instances.

*Does abstraction lead to a pattern?*

So:

**What is an abstraction level in a meta-model, in particular, in our system meta-model?**

Abstraction level in the system meta-model: it is a subset of classes that are…

“An abstraction level L in a metamodel is a subset of classes such that every relationship which cross the frontier have the same orientation and are only inheritance and specialization” [1].

**What defines an abstraction level in our system meta-model?**

More abstract levels (upper levels) have small numbers than concrete ones (lower levels) such as Level 1 is more abstract than level 2.

* Inheritance goes from lower to upper only. (could phrased as all relations go to in one way. This includes inheritance and association). Current system meta-model has all associations abstracted to the most upper level (i.e. level 1).

**How to define abstraction levels in the meta-model? i.e. how to represent these abstraction levels?**

Both meta-models might require adding new classes and operations that are specific to abstraction. for example, in the incident pattern meta-model, we can add a class/interface called *Merge* for *Activity* class and has a method called *canMege()* and *merge()* which define if conditions that an activity needs to meet to merge and also defines how attributes can be merged, respectively.

General methods can be *canAbstract()* and *abstract()* which can be defined over entities of the meta-models. More specialised methods can be defined such as canAbstractContainment() abstractContainment(). Same for connectivity.

Lara et al. [2] defines different abstraction modules such as aggregation, sequential block, similarity, and others.

**When to create new abstraction level (especially with evolving meta-model)?**

* According to Monperrus et al. [1], creating new inheritance relations might indicate that a new level could be needed. However, this is not always the case since one does not need to create a new level for each new inheritance relation. For association relations, it can give a stronger indication that a new abstraction level is needed.

**What would be modified going from one abstraction level to the next?**

Abstraction operations could be included in the meta-models (system and incident pattern) so that we can use them when we create patterns. For example, an abstract() method can be added to the class *Asset* which is then implemented by assets at different levels and provide an abstraction asset based on this implementation. In the incident pattern, also we can have abstraction operations added to the activity, scenes, and maybe others.

**What the output pattern should preserve from the instance (e.g., properties, some qualities )?**

## Abstraction operations

We can define abstraction concepts that can be implemented by the meta-models. These concepts include:

* Merge: several entities are replaced by a single entity the combines the attributes of the other entities.
* Aggregate: add an extra level of abstraction where a new entity is added that contain some entities. The other entities are not replaced they are just restructured under the new entity.
* Delete: remove unnecessary entities in the model. for example remove self-loops, artefacts in a process model (to focus only on the process).

# Abstraction Rules Implemented

**Merging activities based on containment**

A rule is implemented in the incident pattern meta-model to merge a sequence of activities (currently 2 activities are considered).

The rule has the following criteria for two activities to be merged:

* Basic checking: certain conditions need to be met. These are:
  + Have the same action (e.g., “enter”).
  + Have the same initiator (e.g., actor).
  + Have same activity type (e.g., physical)
* In the first activity, the initiator should change its container from the precondition to the postcondition. For example, if in the precondition the initiator is contained in a Hallway, then in the postcondition the initiator should be contained in a different entity.
* In the second activity, the initiator should be contained in a different entity than that in the postcondition of the first activity.
* A connection between the container in the postcondition of the first activity and the container in the postcondition of the second activity.

**Merging activities based on connectivity**

A rule **should be** implemented in the incident pattern meta-model where two activities are merged based on connectivity.

The rule has the following criteria for two activities to be merged:

* Basic checking: certain conditions need to be met. These are:
  + Have the same action (e.g., “enter”).
  + Have the same initiator (e.g., actor).
  + Have same activity type (e.g., physical)
* In the conditions part, the initiator contains an entity which has connections that change (currently increasing) when going from pre to post.
* In the second activity, the same entity (i.e. the one contained in the initiator) in the post condition should have a new connection different from that in the post.

# References

[1] M. Monperrus, A. Beugnard, and J. Champeau, “A definition of ‘abstraction level’ for metamodels,” in *Proceedings of the International Symposium and Workshop on Engineering of Computer Based Systems*, 2009, pp. 315–320.

[2] J. De Lara, E. Guerra, and J. Sánchez Cuadrado, “Reusable abstractions for modeling languages,” *Inf. Syst.*, vol. 38, no. 8, pp. 1128–1149, Nov. 2013.