In order to come from the requirements specifications to the description of the DEECo architecture a software engineer can use the IRM-SA method. The method consists of a model (graph) and a process on how to construct the model. IRM-SA helps in building traceability between system requirements (expressed as goals) and system architecture (expressed as DEECo components and ensembles). While DEECo architectures could be designed by “intuition and common sense”, IRM-SA helps in doing the design systematically, so as all the requirements are well addressed.

IRM-SA modeling concepts in a nutshell

- An invariant is a system goal we want to maintain.
- Invariants can be decomposed into sub-invariants, in AND and OR. OR-decomposition specifies alternative ways to satisfy a parent invariant.
- A leaf invariant is either a process or exchange invariant, or an assumption.
- Process invariants translate 1-to-1 to component processes.
- Exchange invariants translate 1-to-1 to ensembles.
- Assumptions do not translate to processes or ensembles, we use them to validate our design and to model the “switching logic” in case of situation-specific requirements.
- Components contribute their knowledge to the invariants that they take a role in.

Semantics: At runtime, the current view of the situation is determined (this view can be different for different e-cars!) and the system switches to the branch where all the assumptions are true.

Illustration of the IRM-SA modeling concepts on the running example

An effective IRM-SA model (graph) of the ECNP can be considered the following:
**IRM-SA design process**

The design process is a mixed top-down and bottom-up iterative process. You need to:

1. Find the **top-level goals** of the system and specify the top-level (abstract) invariants.
2. Find the **components** of the system by asking “which knowledge does each invariant involve and where is this knowledge obtained from?”
3. Decompose each invariant by asking “how can this invariant be satisfied?”
4. Separate the concerns of the abstract invariants into sub-invariants that correspond to (abstract) **activities that can be done in isolation**.
5. Compose invariants together by asking “why do I need to satisfy these invariants?”
6. In case of situation-specific requirements, try first to accurately capture the **condition** of being in one situation or another. Use the **assumptions** to do that. Then use OR decomposition to specify which invariants to satisfy in each situation.

Once you create a satisfactory IRM model, try to translate each process invariant to a description of a component process and each exchange invariant to a description of an ensemble (as shown in the previous page). If needed, change the graph to capture additional processes and/or ensembles that were overlooked.

**Illustration of the process on the running example**

Consider the application of steps 1-6 to building the IRM-SA model of the ECNP depicted above:

1. There is only one top-level goal of the system: “Up-to-date plan is available”.
2. In order to satisfy the top-level goal we need to specify two components: the “E-Car” component, which will possess the plan, but also the “Parking” component, which will provide input to the (so far abstract) plan by providing information about its availability of parking spaces to the “E-Cars”.
3. In the first iteration, we skip this step (no 3), because we don’t yet see how to decompose the top-level invariant that we specified.
4. According to the provided requirements, each E-Car has to continuously monitor its energy (req. 1a, page 3.2), its position (req. 1b) and to decide whether the existing plan is feasible based on the energy, position and POI (req. 1c). All these requirements correspond to **concrete** activities that can be done in isolation, so for each activity we specify a process invariant (right-most part of the diagram). We can also model the concrete activity of computing a plan (req. 1d) and the concrete activity of keeping an up-to-date availability (req. 2) as process invariants, whereas the concrete interaction between “E-Car” and “Parking” components as an exchange invariant (as all three activities can be done in isolation).
5. By looking at the first three process invariants of the previous step and trying to deduce the reason to specify them in the system, we can conclude that they all together contribute to the more abstract invariant of “checking the feasibility of the plan”, so we add this as their parent invariant. Similarly, we can deduce that the last three leaf invariants specified in the previous step contribute to the more abstract invariant of “having an up-to-date feasible plan with respect to parking availability”, so we add this as their parent invariant.
6. We are interested in the situations where distance of each e-car from its POI is more/less than the threshold of 5 km, so we specify the two assumptions accordingly. As a next step, we **modify the last decomposition of the previous step** to specify an alternative decomposition (different branching according to the situation captured by the assumptions). This integrates also the requested frequency requirements (req. 4 & 5, page 3.2) to the design, i.e., the bottom-most process and exchange invariants are now even more concrete regarding timing requirements (“... within 60 secs”).

Iterating over the steps 2-6, we can now connect the two parent invariants to the top-level invariant (identified in the 1st step) and acquire the final graph depicted on page 3.4.