**Ideas regarding practical implementation**

Difference – for us deep modeling is just a tool to express entity-relationship hierarchies. We are not trying to embed the concept of deep instances into models. We do not want to extend the notion of deep modeling further. Our key approach and difference are that we support variability. That means that we have some snapshots of a big model at runtime and the instance is attached to this snapshot. In other words, in our case, there are two model spaces – model space (defining models) and user space (instances). When an instance has been created the model that is used at this time is forked and related to this instance. That means that if the same model is changed later at some, the instance will stay invariant because it contains enough knowledge to construct its slots, values, and links. There is a lot of inconsistency -> deep instance is not consistent with the current model but consistent with its model. This allows us to achieve variability. Again, we are inconsistent on user-model space but we are consistent in user space.

To make constraints consistent with the current approach it would make sense to define constraints in model space but as soon as an instance is created -> those constraints should be copied to user space. This means that constraints will be also consistent in user space only to its model defined inside of user space.

Constraints should make sense of course. If something is invalid, we should consider either soft or strict constraint evaluation. Soft constraint evaluation means that if some constraint is invalid, we still allow a user to execute an operation but we warn him. Strict constraint evaluation means that we do not allow a user to execute a certain operation until all invalid constraints are fixed by the end user.

Persistence and specification of constraints should be done inside of Codi-framework. My module should only be able to work with the deep model, specify constraints and be able to verify it. So, my module should only serve as an engine to evaluate if a constraint is valid. Therefore, technology should be chosen that can meet the following aforementioned requirements. For this, I have to conduct multiple experiments.  
  
Current technologies in mind:

1. OCL using Eclipse (write a model transformation to convert codi structure into emf structure)
2. Use expression evaluation language (spel, jexl, ognl, mvel)
3. Consider writing your own ontology with OWL
4. Use SHACL as an alternative to OWL
5. Evrete as a rule engine

After the technology is chosen, one must think about a way how to help the end-user to write such constraints and implement them either in your module and embed it in codi or implement it in codi directly.

Apart from it, lambda updates can be considered or ENUM constraints should be considered (in this case the specification of ENUM should be added to codi)

What types of constraints? Attribute, association constraints during instantiation. Constraints on a model level (certain operations should be restricted on a model level if some constraints are defined) E.g., Car must have an attribute built-number if we want to have some other attribute or association.