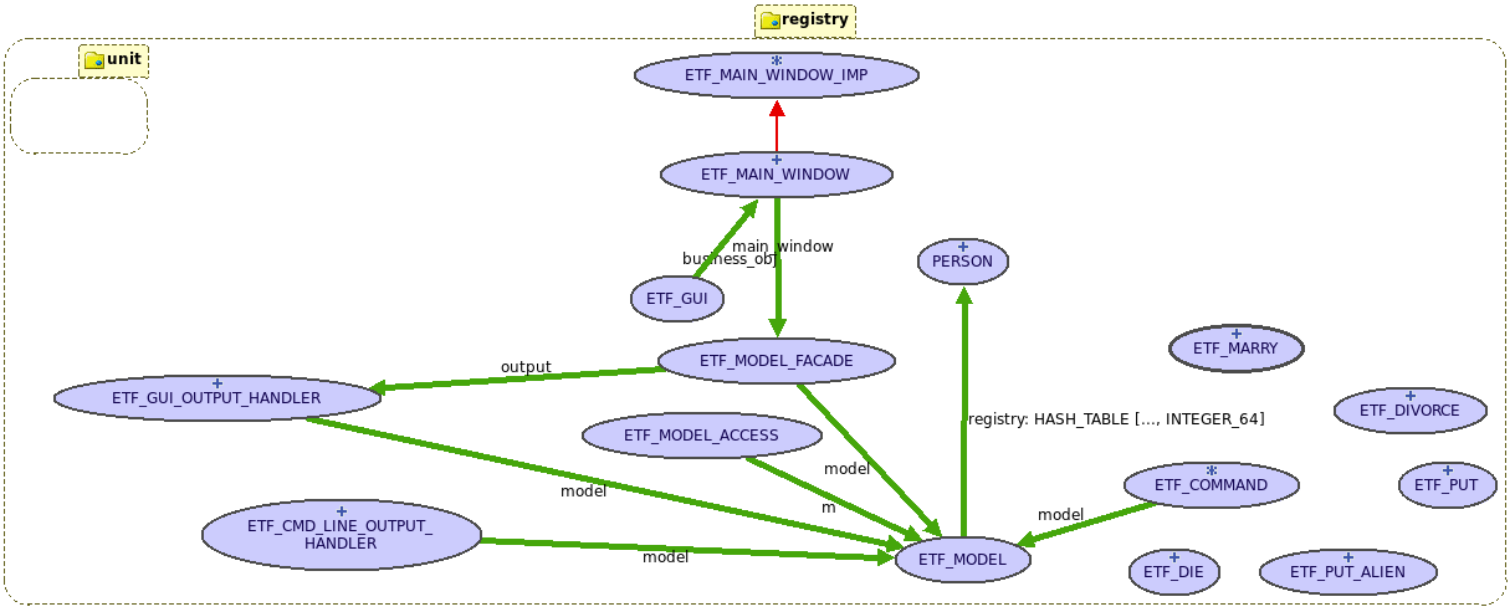
Name: Philip D'Aloia
PRISM login: pdaloia
Student ID: 213672431

Disclaimer:
All work that is submitted in this lab, is that of my own in accordance to York University's policies of academic honesty.

The hardest part of this lab consisted of getting the defensive programming to work properly. Between negating the preconditions from ETF_MODEL and making sure proper error messages were printed, it was a difficult task. To complete all error handling and reporting to match that of the provided oracle, the duration lasted approximately 5-6 hours until all errors were handled as specified.

Part 1: BON Diagram of the Model cluster



To extend the business logic to various types of non-citizens (e.g. visitors, landed immigrants, refugees etc.), I would create new classes for each which would inherit from PERSON. This would let all the new classes have the functionality of a person of type PERSON. In addition to this functionality from PERSON, I would add new attributes that would store the necessary information required for a person which is a type of that new class. For example, for a VISITOR, it would have all information inherited from PERSON

but in combination additional fields would be added such as the amount of time they will be visiting for, arrival date, expected date to leave and then additional features such as extending visiting period and so on.

Part 2: Contract view of the model class

Included on the next page (for proper formatting) is the contract view of the model class PERSON. This was generated by the IDE into a file in RTF form. This was then put into this report.

```
	marry (id1: INTEGER_64; id2: INTEGER_64; date: TUPLE [d: INTEGER_64; m:
INTEGER_64; y: INTEGER_64])
		require
			id_not_same: id1 /= id2
			id_positive: id1 > 0 and id2 > 0
			valid_date: date.y > 1899 and date.y < 3001
			valid_month_and_day: check_valid_date (date)
			id_unused: registry.has (id1) and registry.has (id2)
			valid_age: valid_marriage_date (date, id1) and valid_marriage_date
(date, id2)
			not_dead: not check_if_dead (id1) and not check_if_dead (id2)
```

This is the part of the model which deals with marriages. These are the preconditions necessary to ensure that a “marry” operation can be valid.