Documentation for the Social Practice Agent (SoPrA) Model

Rijk Mercuur

August 13, 2019

1 Overview

Netlogo There are three Netlogo files:

SoPrANetlogoTemplate Captures the UML by implementing classes and assocations as breeds and links. Can be used as the basis for an ABM.

SoPrANetlogoSetupExample Shows how to use set-up the model for a use case on commuting by using several input files.

SoPrANetlogoFullModelExample Shows how SoPrA enables a ABS researcher to study habits. This version contains both a setup and a decision-making algorithm.

Repast The Java Repast version of SoPrA is split into abstract classes and domain-specific implementations of those classes.

Protege The Protege version of SoPrA has two files. sopra.owl fully corresponds to the UML whereas soprasimple.owl is a slight simplification that greatly diminishes the complexity.

2 UML Model

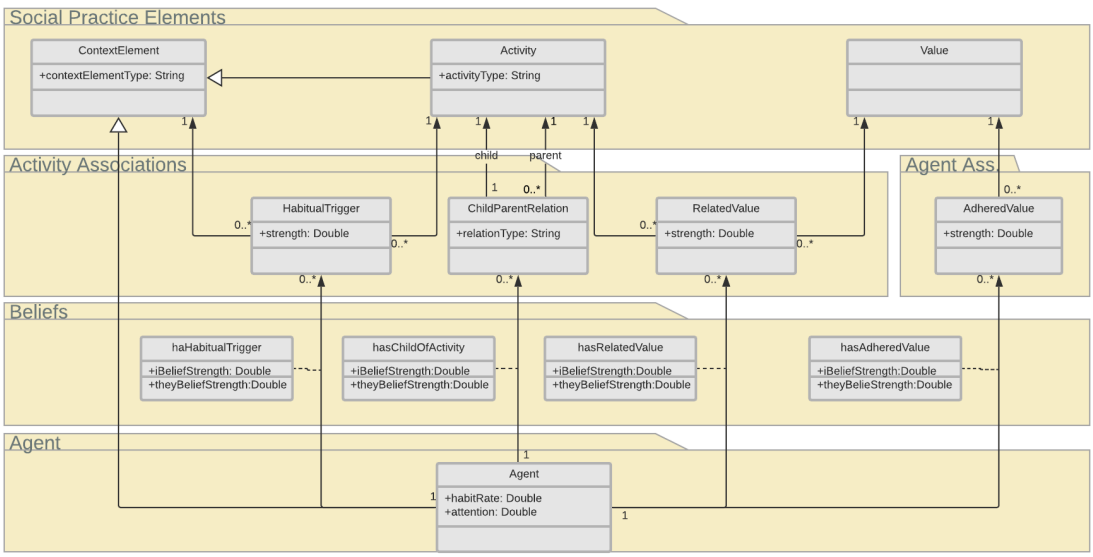

Figure 1: The SoPrA UML depicting packages (yellow envelops), classes (grey boxes), association classes (grey boxes attached to an association with a dotted line) and associations (arrows).

3 Implementation Choices

Repast Simphony Repast Simphony is a java-based tool for programming ABS based on the Eclipse IDE (**?**). The translation from the UML to java code is relatively straight-forward as both are based on the object-orientated design method. An implementation choice will have to be made regarding the implementation of association classes. Let us take the `hasHabitualTrigger` assocation class as an example. The implementation strategy that corresponds most clearly to the UML is to create a separate class `HasHabitualTrigger`, a variable in this class that links to the `HabitualTrigger` class (e.g., `HabitualTrigger myHabitualTrigger`) and a list in the agent class that links to the `HasHabitualTrigger` class (e.g., `List<HasHabitualTrigger> myHasHabitualTriggers`). Another implementation strategy that produces less code, but less clearly corresponds to the UML is to create a table in the agent that maps every `ContextElement` and `Activity` pair to a triple of three doubles representing the `habitualTriggerStrength`, the `iBelieftStrength` and the `theyBeliefStrength` (e.g., `Table<ContextElement, Activity, StrengthValues<Doubl`
The implementation we provide on Github uses the latter method and thus chooses code simplicity over a clear relation between UML and code.

Netlogo Netlogo is a tool for programming ABSs that (compared to Repast) trades part of its programming flexibility for an easy to use high-level programming language and a user-friendly interface (e.g., sliders, visualization) (**?**). Researchers using Netlogo can implement classes as breeds and associations as links. Netlogo has a natural way to implement association classes as links with variables. Note that the SoPrA UML is designed to have only binary associations to make the translation to links in Netlogo (that can only be binary) relatively easy. Netlogo has no default implementation to model inheritance between classes (e.g., the fact that all activities are context-elements). We chose to model inheritance with a reporter (i.e., a function with a return value) that, when called, gives back the intersection of the `Activity` breed and `Agent` breed (and a `Resource` breed and `Location` breed when these are added to SoPrA).

Protégé Protégé is a tool to author formal ontologies and enable formal reasoners that aid the modeller (**?**). Although SoPrA is not agent-based programming tool it can help researchers to check if they made a proper instantiation of SoPrA for their use case. In other words, modellers can use Protégé to make domain-specific SoPrA model and use the formal reasoner to check if this corresponds to the general SoPrA ontology we made. Protégé makes this possible by its use of description logic: a logic that trades its expressivity to acquire decidability (**?**). The UML classes correspond to classes in Protégé and UML associations to object properties. **?** shows how to capture association classes and formal rules regarding value inheritance in Protégé. Consequently, this shows that description logic is expressive enough to capture the semantics of SoPrA and that the formalization of SoPrA in Protégé is satisfiable.