J. Gary Polhill (2015)

Extracting OWL Ontologies from Agent-Based Models: A Netlogo Extension

Journal of Artificial Societies and Social Simulation 18 (2) 15
<http://jasss.soc.surrey.ac.uk/18/2/15.html>

Received: 09-Oct-2014 Accepted: 24-Feb-2015 Published: 31-Mar-2015

Abstract

Using OWL ontologies to represent the state and structure of a simulation at any one time has been argued to improve the transparency of a social simulation, on the basis that this information is then not embedded in the source code of the model, or in the computer's memory at run-time. Should transparency of such a form be desirable, it would be preferable to enable it by extracting the information automatically from a running model. However, semantic differences between traditional object-oriented programming languages and description logics pose an obstacle to this. This paper presents arguments that Netlogo does not have the same semantic challenges to automated ontology extraction, and describes an extension to Netlogo (5.0) using the OWL-API (3.1.0) that extracts state and structure ontologies from an existing Netlogo model. The extension is freely available from https://github.com/garypolhill/netlogo-owl.

Keywords:
Netlogo, OWL, OWL-API, Ontology, Transparency

Introduction

1.1 Social simulation is often argued to be more transparent than other approaches to modelling social systems (e.g.Galán et al. 2009). The transparency of a social simulation can refer to the ability to relate the state and structure of a simulation to those of the target system it is intended to represent (e.g. Leydesdorff 2001), but also to the relationship between expressions in the source code and properties of the target system (e.g. Taylor 2003). In general, transparency in this context may be conceived as the ability of someone other than those immediately involved in a model's construction to understand what it is doing (e.g. Polhill & Gotts 2009). However, as Janssen et al. (2008) point out, whilst full transparency is ideal, 'translucency' – in which there is a partial exposure of the implementation – may often be all that is practical. Indeed, sometimes the latter may be preferred; for example, when discussing certain aspects of the model in inter- or trans-disciplinary teams.

1.2 Whilst computer programming languages can be reasonably argued to efficiently and formally describe algorithms, extracting the structure of a simulation (the classes of entity in the model, the properties they have, and the relationships they have with other entities) from its source code can be challenging. In an object-oriented model, the structure may be distributed across several class files, with no immediately obvious way to determine which classes are ontologically significant, and which are there to provide functionality not already provided by the language API or any programming libraries used. Although extracting UML diagrams from object-oriented code can be automated, there is the further problem that inheritance in object-oriented models reflects compiler and programmer convenience, rather than common-sense understandings (LaLonde & Pugh 1991). If there is an acceptance that logics are formalisations of common-sense understandings, then for the purposes of transparency (or translucency) rather than programming, the ideal would be that the model structure is represented using axioms of description logics.

1.3 Semantic differences between classical object-oriented programming languages and description logics have been highlighted by Polhill and Gotts (2009), with reference to Lalonde and Pugh's (1991) distinctions among the three inheritance relations subtype, subclass and is-a. Of these three, the first is for compiler convenience in checking types involved in operations (`interfaces` in Java), the second for programmer convenience in facilitating code re-use (`classes` in Java), whilst the third reflects descriptive, logical meanings that are of primary interest from an ontological point of view. These differences mean that direct ontology learning from model source code is not guaranteed in object-oriented programming languages.

1.4 Netlogo (Wilensky 1999) models do not have these issues because Netlogo is not an object-oriented programming language, and there is no inheritance among the language's main entity ('agent' in Netlogo terminology) types, which are known as 'breeds'. The facility for explicit representation of relationships (or 'links') among agents, rather than encapsulating associations in classes as in object-oriented languages, further contributes to enabling automatic ontology learning from Netlogo model code, though, as discussed later, doing so imposes a particular programming style on the implementation.

1.5 Netlogo's extension facility enables software to be written that provides functionality from within Netlogo to automatically extract an OWL (Horrocks et al. 2003) or OWL 2 (Cuenca Grau et al. 2008) ontology from the model. OWL is a popularly used language for expressing ontologies, which has correspondences with description logics that depend on the sublanguage used. The mapping is shown in Table 1.

Table 1: Mapping from Netlogo syntax to OWL ontology types.

Netlogo syntax	OWL type(s)	Comments
`breed`	Class	Breeds in Netlogo, like classes in OWL, are not necessarily disjoint – agents can change breed using the `set breed` command.
`directed-link-breed` (no '`own`')	ObjectProperty	For all link breeds, there is no syntax to constrain the breeds of agent that are at either end of the link, meaning that domains and ranges for links cannot be inferred from the code. If the link breed does not have any of its 'own' variables, it can be declared directly as an object property…
`directed-link-breed` (with '`own`')	Class and ObjectProperty	…otherwise the link must be reified, and two object properties used to link the breeds that can be connected in this way. The object property going in to the class representing the property can be declared inverse functional; the object property going out can be declared functional. A property chain can be used to represent the link as a whole if the ontology is in OWL 2.
`undirected-link-breed`	ObjectProperty or Class and ObjectProperty	In addition to the points applying to directed link breeds, object properties corresponding to undirected link breeds (in the case of reified links, this will apply to the chain) can be declared symmetric in OWL. If the ontology is in OWL 2, all links can be declared as irreflexive as Netlogo does not permit an agent to link to itself.
`breeds``-own`	DataProperty	The domain of the data property can be declared to be the class corresponding to the breed. However, since Netlogo does not have typed variables, the data range cannot be automatically inferred. Note that different breeds can have the same variable.
`link-breeds``-own`	DataProperty	The domain of the data property can be declared to be the class corresponding to the link breed. Note that different link breeds can have the same variable, but a variable cannot be shared between link breeds and breeds.
agent	Individual	Agents in Netlogo can be asserted as individuals, and members of the class corresponding to the breed of which they are currently a member.

1.6 In the rest of this paper, an extension to Netlogo implementing these mappings is presented, together with an illustration based on the CEDSS (Community Energy Demand Social Simulator) model, built as part of the GILDED (Governance, Infrastructure, Lifestyle Dynamics and Energy Demand) Framework Programme 7 project (http://gildedeu.hutton.ac.uk/).

The Netlogo Extension

2.1 The `owl` extension to Netlogo was written to implement the mappings in Table 1. The extension is written for Netlogo 5.0, and consists of the file owl.jar. This, together with version 3.1.0 of the OWL-API (Bechhofer et al. 2003; Horridge et al. 2007; http://owlapi.sourceforge.net/), which is available under the GNU Library General Public Licence, should be placed in a subdirectory named 'owl' of the directory in which the .nlogo file using the owl extension is located. The file 'owl.zip' – the release version of the extension, contains owl.jar and the OWL-API (owlapi-bin.jar), ready for use. To permanently install the extension, put it in the extensions folder of the Netlogo applications directory.

2.2 The extension makes available a number of commands to a Netlogo model containing the following command (typically, near the beginning of the code):

```
extensions [owl]
```

The commands are given in approximate order in which they must occur during execution of the model. Specifically,

`owl:domain`, `owl:range` and `owl:imports` cannot be used after `owl:model`, and `owl:structure` and `owl:state` may only be used after `owl:model`.

2.3 The commands assume a distinction between the *structure* and the *state* of the model and corresponding ontologies. An ontology describing the structure of a model consists of terminology (T-box) axioms describing the kinds of entity that the model contains (OWL classes), properties they have (OWL data properties) and relationships they may have with other types of entity (OWL object properties). By contrast, an ontology describing the state of a model applies to a particular snapshot of an instance of it running at one time. The state ontology imports the structure ontology, and then adds assertion (A-box) axioms about the specific instances of the various classes of entity that exist in the model when the snapshot is taken.

`owl:domain` *`link-breed breed`*

2.4 The `owl:domain` command takes two arguments. The first is the (string) name of a link-breed, and the second is the (string) name of a breed. The command causes an assertion to be made in the ontology that the OWL class corresponding to the breed is in the domain of the OWL object property corresponding to the link-breed.

2.5 This command will not affect your model (that is to say, while the model runs, no check will be made of the link-breed that only agents of the specified breed are in the domain of the link-breed).

`owl:range` *`link-breed breed`*

2.6 The `owl:range` command takes two arguments. The first is the (string) name of a link-breed, and the second is the (string) name of a breed. The command causes an assertion to be made in the ontology that the OWL class corresponding to the breed is in the range of the OWL object property corresponding to the link-breed.

2.7 Just as for `owl:domain`, `owl:range` does not cause any checks to be made while the model runs that only agents of the specified breed are in the range of the link-breed.

`owl:imports` *`IRI`*…

2.8 The argument(s) to the `owl:imports` command are strings containing Internationalised Resource Identifiers for ontologies that the model structure ontology is to import. Again, this does not affect the model in any way, or cause any checks to be made, but allows the ontology to be created with the imports in place ready for use with ontology visualisation, reasoning and analysis tools.

`owl:model` *`IRI`*

2.9 Define the Internationalised Resource Identifier for the model structure ontology, which is contained in the string argument. All ontologies have an IRI, which constitutes an identifier for the ontology. Although the IRI is usually a web address, it is not necessarily the case that the file will exist at that web address (though ideally it would) – the IRI is just a name.

2.10 As stated above, all `owl:domain`, `owl:range` and `owl:imports` commands must execute before `owl:model` is executed, and no `owl:structure` or `owl:state` command may execute until `owl:model` has executed.

`owl:options` *`option-string`*…

2.11 Set options for creating structure and state ontologies. Three option strings are available:

- "`owl2`", which will add OWL 2 axioms to the ontology where relevant;
- "`relations`", which will add axioms asserting relationship attributes (e.g. symmetric, and if "`owl2`" is specified, irreflexive and asymmetric) to property assertions in the ontology where relevant; and
- "`no-patches`", which suppresses spatial axioms. Spatial axioms include the assertion of a `location` object property with range `Patch`, and hence the declaration of the latter as a class.

The option "`none`" (the default) can be used to indicate that none of the options above applies. The `owl:options` command can be run more than once, overriding any options selected in earlier calls; subsequent calls to `owl:structure` and `owl:state` will respect the options selected.

`owl:structure` *`file-name`*

2.12 Save a model structure ontology to the file name given as arguments. The logical IRI of the ontology will be that specified using the `owl:model` command. Note that for visualisation software such as OntoViz to show a link as an object property in the resulting diagram, both `owl:domain` and `owl:range` must have been specified. (The software would otherwise not know which classes to draw the link from and to respectively.)

`owl:state` *`file-name time-step`* (experimental)

2.13 Save an ontology of the current state of the model to the file name. The logical IRI for the state ontology will be constructed from the model IRI specified by the earlier `owl:model` command and the time-step given as second argument to this command. Specifically, any .owl suffix will be removed from the model IRI, and then "`-state-`" appended, followed by the time-step, followed by the extension string if given, followed by the .owl suffix if the model IRI had it originally.

2.14 For example, the model IRI "http://www.gildedeu.org/ontologies/CEDSS-3.3.owl" given the argument 7 for the time-step, would produce a state ontology IRI "http://www.gildedeu.org/ontologies/CEDSS-3.3-state-7.owl". (Note that the links have different targets from those shown.)

2.15 The `file-name` argument gives a physical location to save the ontology to. If it does not end with "`.owl`", then the name of the resource in the generated state ontology IRI is appended to the `file-name` argument to produce the final location. For example, with the state ontology IRI above (if `ticks` = 7):

```
owl:state "/var/tmp/" ticks
```

will save the ontology to `/var/tmp/CEDSS-3.3-state-7.owl`.

2.16 An exception is thrown if an attempt is made to overwrite an existing file.

2.17 The `owl:state` command can be quite demanding on resources depending on the size of the model. As well as the file saved being potentially quite large in size, the command can take several minutes to search through all the elements in the model, and may crash with heap errors if Netlogo is not run with sufficient memory given to the Java virtual machine. (The example in Figure 3 below takes 1 minute or so on a machine with a 2.8GHz Intel Core i7 processor, requires 8G of RAM for the Java virtual machine, and the resulting OWL file (in RDF/XML format) occupies 650M or so of disk space if not compressed.)

Illustration

3.1 The CEDSS (Community Energy Demand Social Simulator) model was built to simulate domestic energy use patterns in a small community (e.g. a housing estate or a village), and is based on an earlier prototype model with the acronym ABMED (Agent Based Model of Energy Demand; Gotts 2009). It represents households as agents, who live in dwellings, and have a portfolio of appliances they use for space and water heating, and entertainment. Using a representation of the psychological theory of 'goal frames' (Lindenberg & Steg 2007), in which consumer decisions are based around different modes of behaviour (in this case, hedonistic, egoistic, and biospheric), CEDSS agents decide how to replace broken appliances, and whether to buy any new appliances. Interactions are based around agents visiting other houses in their social network. Visitor agents observe and potentially desire appliances they don't own that their hosts possess, and adjust goal frame parameters to make the probabilities of different modes of behaviour more similar to those of their hosts.

3.2 The following procedure demonstrates the use of some of the above ontology features to save a structure ontology from CEDSS:

```
to save-structure [file-name]
  owl:domain "social-links" "households"
  owl:range "social-links" "households"
  owl:domain "replacements" "appliances"
  owl:range "replacements" "appliances"
  owl:domain "addresses" "households"
  owl:range "addresses" "dwellings"
  owl:domain "insulates" "insulations"
  owl:range "insulates" "dwellings"
  owl:domain "ownerships" "households"
  owl:range "ownerships" "appliances"
  owl:domain "consumes" "appliances"
  owl:range "consumes" "consumption-patterns"
  owl:domain "uses" "consumption-patterns"
  owl:range "uses" "fuels"
  owl:domain "upgrades" "insulations"
  owl:range "upgrades" "insulations"
  owl:model "http://www.gildedeu.org/ontologies/CEDSS-3.3.owl"
  owl:structure file-name
end
```

Figure 1. Example procedure to extract a structure ontology from CEDSS.

3.3 Using the OntoViz (Sintek 2007) plug-in for Protégé 3, a DOT graph can be created of the resulting ontology, which (with some minor editing to simplify the text, enter datatypes and highlight reified link-breeds in blue) is shown in Figure 2.

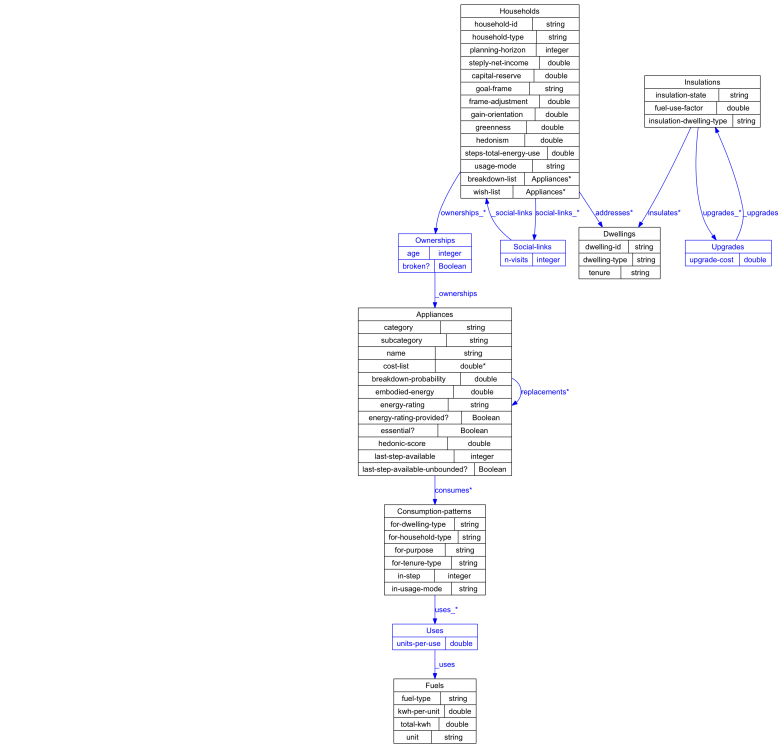


Figure 2. Ontology extracted from CEDSS visualised with OntoViz.

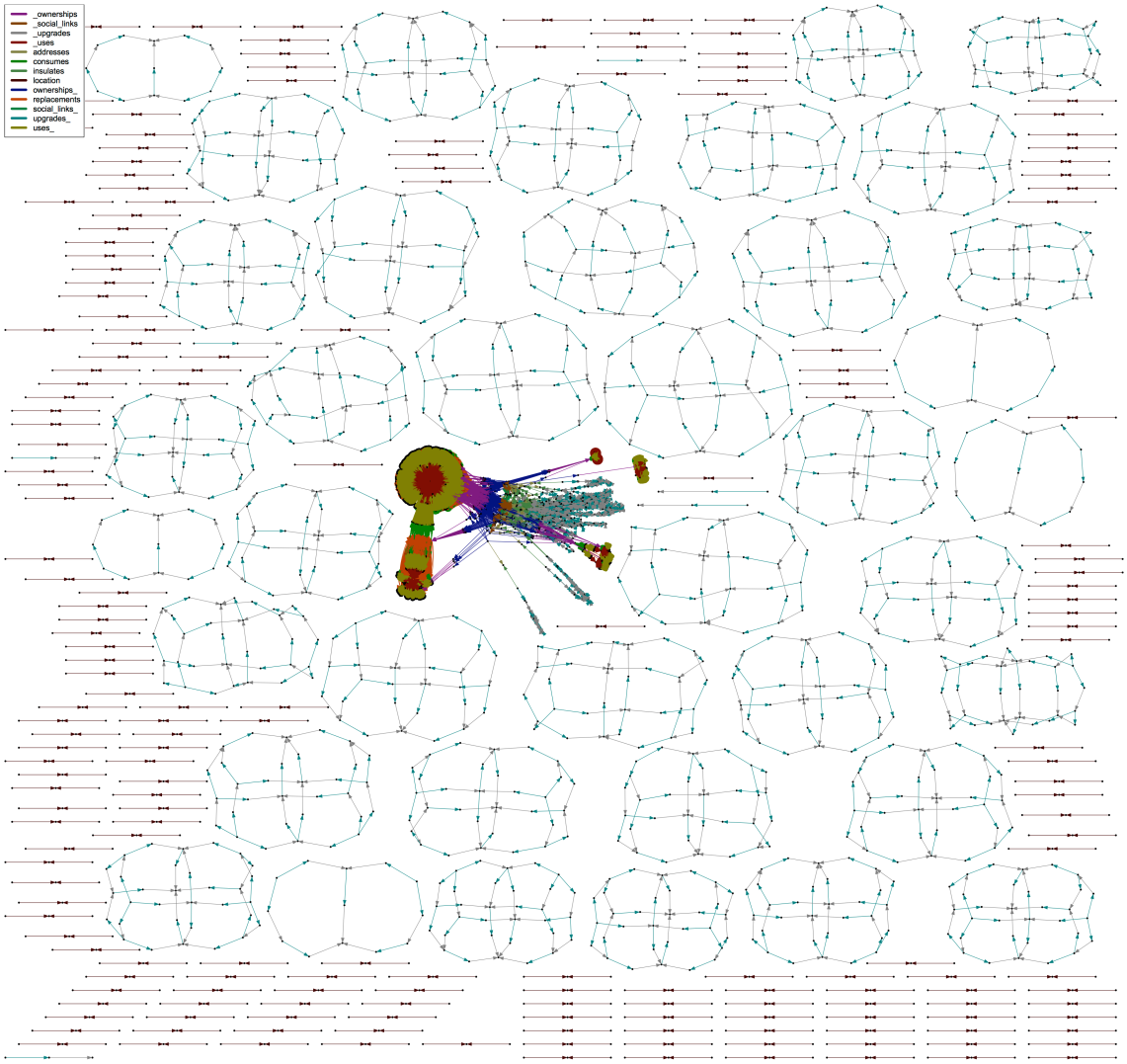


Figure 3. Diagram of all the individuals in a state ontology in the CEDSS model, rendered using the `sfdp` command supplied with GraphViz[1] and a bespoke program to create an appropriately formatted `dot` graph from the ontology. Nodes (black dots) are individuals, coloured arcs are object properties. Apart from the dense graph in the middle, the other arcs show `upgrade`

paths for insulations, and `location` (built-in relationship agents have with patches, which would not have been included if the `no-patches` option had been specified).

Discussion and conclusion

4.1 Whilst the ontology extracted from the Netlogo code makes semantic sense, writing a new model with a view to using the extension may affect the way certain concepts are represented. The treatment of all breeds' '`own`' variables as data properties means that any such variables that contain lists or sets of other agents will not be represented as a relationship in the resulting structure ontology, with the extension as presently written. The effect would be to encourage the explicit representation of relationships using link-breeds. Mapping from breeds to OWL classes would also encourage model developers to make greater use of breeds for model entities than otherwise they might. An 'agent' might normally be thought of as something that undertakes an action, a conceptualisation that tends to oppose the representation of inanimate entities as agents. This may be undesirable from some theoretical perspectives if the concept of an agent in Netlogo is to be taken literally (a view that would arguably be somewhat 'fundamentalist' in character), but is not without theoretical precedent (Dennett 1971; Latour 2005).

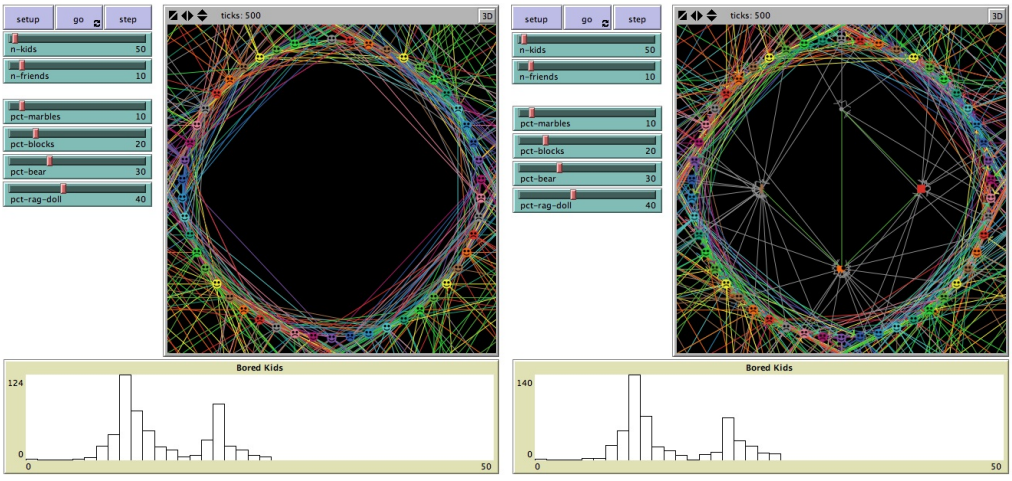
4.2 The other side of this coin is that different coding styles will influence the ontology extracted. Suppose two people use Netlogo to reimplement the same model. Both reimplementations behave in the same way (or with a degree of similarity corresponding to one of Wilensky and Rand's (2007) levels), but the different approach each has used to represent the model using Netlogo datastructures and features means that the ontologies extracted from each implementation will be different.

4.3 Here we use the simple example of a model in which a group of children are playing with toys. Each child has a toy, and, depending on which toy they have will get bored with it after a specified number of time steps and try to swap their toy with that of one of their friends who is also bored. There are constraints on which toys can be swapped (see Table 2). The number of time steps for which each child has the toy they are playing with is recorded, and is reset to zero after each swap. The parameters of the model are the percentages of children having each toy initially, the number of children, and the number of friends each child has (which is the same for all children). We are interested in the distribution of the number of bored children over the time of the model run, which is a histogram of the number of time steps for which there were *x* bored children.

Table 2: Table of toy parameters

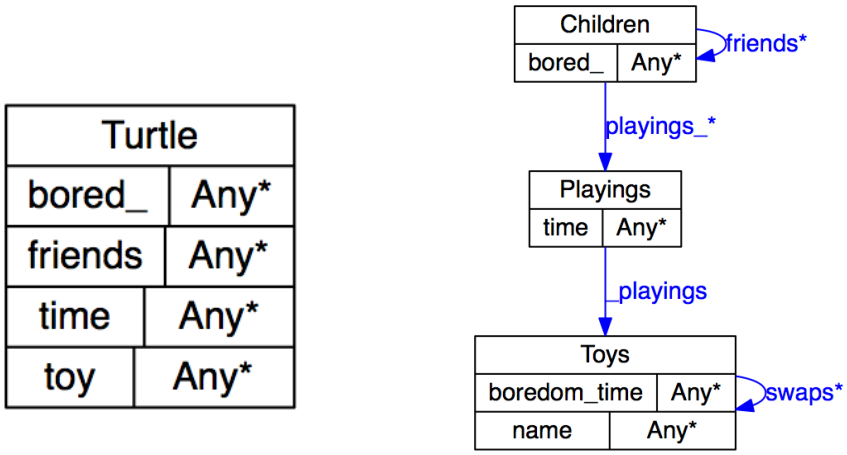
Toy	Time until bored	Can swap with…
marbles	3	blocks, bear
blocks	4	marbles, bear
bear	2	marbles, blocks, rag-doll
rag-doll	1	bear

4.4 There are various ways the model could be implemented, but to highlight the potential differences in coding style, we choose one approach making no use of `breeds` or `links`, and contrast it with a coding style more suitable for use with the `owl` extension. The two implementations have slightly different output after 500 time steps – exact numerical similarity, despite controlling for seed, has not been achieved, but the distributions of numbers of bored children over time are very similar (Figure 4). In Figure 5, however, the difference between the ontologies the extension is able to extract from the two implementations is clear.



(a) (b)

Figure 4. Screenshots from the two implementations using the same parameters. (a) Implementation not using `breeds` or `links`; (b) implementation suitable for `owl` extension (which is also able to show who is playing with which toy). After 500 time steps, the distributions of bored children shown in the histograms are similar, but there are slight differences.



(a) (b)

Figure 5. Difference in ontology generated by the `owl` extension for the two implementations. (b) Shows the ontology from the implementation designed to work with the `owl` extension. Note that in contrast to Figure 2, the reified relationship `Playings` has not been coloured, and datatypes have not been manually entered. This shows more clearly what the owl extension provides by default.

4.5 As Table 3 shows, there are considerable differences in the way the representation in the two implementations is set up. The left column representation just uses the default `turtle` breed and assigns it some variables to store the information used by the model: a list of friends, which toy they are playing with, whether or not they are bored, and the length of time they have been playing with the current toy. In the right hand column, the code is more explicit about the microworld: there are `breed`s for toys and children, link breeds for who is a friend of whom, who is playing with what toy, and which toys can be swapped with which others. As a result, the `children` breed has only the Boolean storing whether or not they are bored. The right hand representation also contains additional information about the toys, with `boredom-time` storing how long it takes for a child to get bored with playing with it. The code in Table 4 shows that the strict implementation not using a`breed` to represent the toys has meant that this information is hard-coded into the model, though there could be ways round this using, for example, the `table` extension in Netlogo to associate the name of the toy with the time.

Table 3: Contrasting representations in the two implementations of the 'toy' model

No breeds or links	Version suitable for **`owl`** extension

turtles-own [
  friends
  toy
  bored?
  time
]
	breed [toys toy]
	toys-own [
	  name
	  boredom-time
	]
	
	breed [children child]
	children-own [
	  bored?
	]
	
	directed-link-breed [friends friend]
	
	directed-link-breed [playings playing]
	playings-own [
	  time
	]
	
	undirected-link-breed [swaps swap]

Table 4: Contrasting implementation of determining whether children are bored

No breeds or links	Version suitable for `owl` extension

ask turtles [
  set time time + 1
  ifelse toy = "marbles" and time > 3 [
  set bored? true
  ]
  [
  ifelse toy = "blocks" and time > 4 [
  set bored? true
  ]
  [
  ifelse toy = "bear" and time > 2 [
  set bored? true
  ]
  [
  if toy = "rag-doll" and time > 1 [
  set bored? true
  ]
  ]
  ]
  ]
]
	ask children [
	  ask my-out-playings [
	  set time time + 1
	  if time > [boredom-time] of other-end [
	  ask myself [set bored? true]
	  ]
	  ]
	]

4.6 Table 4 also shows that there can be some awkward syntax in the coding style used to make the representation in the model more suitable for the `owl` extension. Since Netlogo doesn't have syntax for functional relationships, `my-out-playings` makes it unclear that each child can only play with one toy at a time. Arguably, for those not familiar with Netlogo, the code on the left is clearer about the conditions under which children (even if they are called turtles here) become bored. On the other hand it makes no reference to 'playing' or 'boredom-time', meaning that for those with an understanding of Netlogo syntax, the code on the right is (functional relationship question aside) arguably more transparent in that the hardcoded numbers are replaced with a variable describing what they are supposed to mean.

4.7 Although the fact that these two functionally similar implementations have different ontologies could be seen as a weakness, it could equally be argued that this is a reflection of variations in transparency of coding style. That is, two functionally similar implementations can still be compared on the basis of how clearly their source code reflects the system they are representing. Whether this extension to Netlogo has any role to play in making such a judgement is debatable, if, indeed, such questions are anything more than a matter of subjective preferences. Insofar as there are objective measures of transparency (an example of which could be the ease with which inferences can be made using information about the model state or structure), software such as the Netlogo extension presented here could prove useful.

Availability and Requirements

The software is available on GitHub under a GNU Lesser General Public Licence at https://github.com/garypolhill/netlogo-owl.

It requires the OWL-API version 3.1.0 available from SourceForge at http://sourceforge.net/projects/owlapi/files/OWL API (for OWL 2.0)/3.1.0/.

Acknowledgements

This work was funded by the European Commission Seventh Framework Programme, grant agreement number 225383, and the Scottish Government Rural Affairs and the Environment Portfolio Strategic Research Theme 4 (Economic Adaptation).

Notes

[1]http://www.graphviz.org/

References

BECHHOFER, S., Volz, R. & Lord, P. (2003). Cooking the semantic web with the OWL API. In Fensel, D., Sycara, K. & Mylopoulos, J. (eds.) *Second International Semantic Web Conference, ISWC 2003, Sanibel Island, Florida, 20–23 October 2003. Lecture Notes in Computer Science* 2870, 659–675. [doi:10.1007/978-3-540-39718-2_42]

CUENCA GRAU, B., Horrocks, I., Motik, B., Parsia, B., Patel-Schneider, P. & Sattler, U. (2008). OWL 2: The next step for OWL. *Journal of Web Semantics* 6, 309–322. [doi:10.1016/j.websem.2008.05.001]

DENNETT, D. C. (1971). Intentional systems. *The Journal of Philosophy* 68 (4), 87–106. [doi:10.2307/2025382]

GALÁN, J. M., Izquierdo, L. R., Izquierdo, S. S., Santos, J. I., del Olmo, R., López-Paredes, A. & Edmonds, B. (2009). Errors and artefacts in agent-based modelling. *Journal of Artificial Societies and Social Simulation* 12 (1), 1: http://jasss.soc.surrey.ac.uk/12/1/1.html

GOTTS, N. (2009). ABMED: A prototype model of energy demand. *Sixth Conference of the European Social Simulation Association, University of Surrey, Guildford, Surrey, 14–18 September 2009*.

HORRIDGE, M., Bechhofer, S. & Noppens, O. (2007). Igniting the OWL 1.1 touch paper: The OWL API. *Proceedings of the Third OWL Experience and Directions Workshop, OWLED 2007, Innsbruck, Austria, June 2007*.

HORROCKS, I., Patel-Schenider, P. F. & van Harmelen, F. (2003). From SHIQ and RDF to OWL: The making of a Web Ontology Language. *Journal of Web Semantics* 1, 7–26. [doi:10.1016/j.websem.2003.07.001]

JANSSEN, M., Alessa, L. N., Barton, M., Bergin, S. & Lee, A. (2008). Towards a community framework for agent-based modelling. *Journal of Artificial Societies and Social Simulation* 11 (2), 6: http://jasss.soc.surrey.ac.uk/11/2/6.html

LALONDE, W. & Pu, J. (1991). Subclassing ≠ subtyping ≠ is-a. *Journal of Object-Oriented Programming* 3 (5), 57–62.

LATOUR, B. (2005). *Reassembling the Social – An Introduction to Actor-Network-Theory.* Oxford University Press.

LEYDESDORFF, L. (2001). Technology and culture: The dissemination and the potential 'lock-in' of new technologies. *Journal of Artificial Societies and Social Simulation* 4 (3), 5: http://jasss.soc.surrey.ac.uk/4/3/5.html

LINDENBERG, S. & Steg, L. (2007). Normative, gain and hedonic goal frames guiding environmental behavior. *Journal of Social Issues*, 63(1), 117-137. [doi:10.1111/j.1540-4560.2007.00499.x]

POLHILL, J. G. & Gotts, N. M. (2009). Ontologies for transparent integrated human-natural systems modelling. *Landscape Ecology* 24 (9), 1255–1267. [doi:10.1007/s10980-009-9381-5]

SINTEK, W. (2007). OntoViz. Available from http://protegewiki.stanford.edu/index.php/OntoViz.

TAYLOR, R. I. (2003). Agent-based modelling incorporating qualitative and quantitative methods: A case study investigating the impact of e-commerce upon the value change. Ph. D. Thesis, Centre for Policy Modelling, Manchester Metropolitan University. CPM Report No. CPM-03-137. http://cfpm.org/cpmrep137.html

WILENSKY, U. (1999). *Netlogo*, http://ccl.northwestern.edu/netlogo/. Center for Connected Learning and Computer-Based Modelling, Northwestern University, Evanston, IL.

WILENSKY, U. & Rand, W. (2007). Making models match: Replicating an agent-based model. *Journal of Artificial Societies and Social Simulation* 10 (4), 2: http://jasss.soc.surrey.ac.uk/10/4/2.html