Generating Executable Code from High-level Formal Description of Social or Socio-Ecological Models*

Themis Dimitra Xanthopoulou[1][0000−0003−2914−0472], Andreas Prinz[1][1111−2222−3333−4444], and Fount LeRon Shults[2][2222−−3333−4444−5555]

[1] University of Agder, Jon Lilletuns vei 9, 4879 Grimstad, Norway
themis.d.xanthopoulou@uia.no
https://www.uia.no/en/kk/profil/themisdx
[2] University of Agder, Universitetsveien 19, 4630 Kristiansand, Norway

Abstract. Agent-Based Modelling has been used for social simulation because of the several benefits it entails, including the capacity to improve conceptual clarity, enhance scientific understanding of complex phenomena, and contribute policy-relevant insights through simulation experiments. Social models are often constructed by inter-disciplinary teams that include subject-matter experts with no programming skills. These experts are typically involved in the creation of the conceptual model, but not the verification or validation of the simulation model. The Overview, Design concepts, and Details (ODD) protocol has emerged as a way of presenting a model at a high level of abstraction and as an effort towards reproducibility of Agent Based Models (ABMs). However, this popular protocol does not improve the involvement of experts because it is typically written after a model has been completed. This paper reverses the process and provides non-programming experts with a user-friendly and extensible tool called ODD2ABM for creating and altering models on their own. This is done by formalizing ODD using concepts abstracted from the NetLogo language, enabling users to generate NetLogo code from an ODD description automatically. We verified the ODD2ABM tool with three existing NetLogo models and assessed it with criteria developed by other researchers.

Keywords: Social model · Meta-model · Code generation · Abstraction · Formality · Reproducibility · Verification .

1 Introduction

In recent years there has been a rapid rise in the use of the Agent-Based Modelling [2], which typically involves micro-simulation based on individually (inter)acting sub-models [29]. Distinct from the more generic Multi-Agent Models,

* Supported by the University of Agder

ABMs offer unique capabilities and are widely used by a growing number of scientists and policy professionals [5, 10, 11, 21]. One of the prominent applications of ABMs is in social simulations.

ABMs contribute to conceptual clarity for the ambiguous concepts in social science. For example, most of us immediately grasp the general meaning of the concept of bullying, but it can be difficult to determine precisely the sort of incident that falls into that category. Definitions of bullying that are too vague make it difficult to identify and mitigate the relevant behaviours. The creation of an ABM requires scientists to clarify vague concepts by explicitly linking the concepts to agent behaviours, agent properties and interactions, which are easier to compare to real life situations and more likely to render policy-relevant insights.

Hassam et al. have identified four roles in the development of an ABM: the thematician, the modeller, the computer scientist and the programmer [11]. In the following section we will use the paradigms of the model development process presented in [22] to explain the function of each role. A model is the representation of a real-life "problem entity". The *thematician*, who is the expert in the problem entity, is the person who creates the first conceptual model of the problem entity by providing the context of the ABM, system theories and expert knowledge. The *modeller* transforms the description of the first conceptual model into a more formal conceptual model or a simulation model specification, the *computer scientist* finds an executable approximation, and the *programmer* implements the simulation model using a selected programming language and platform. Rarely can one researcher cover all these roles. Moreover, complex models usually require perspectives from different disciplines and therefore more than one thematicians. These are the reasons why ABMs are typically constructed by multidisciplinary teams.

These teams face several challenges. First of all, communication of the conceptual model among such diverse researchers can be difficult [11, 21, 23]. Since the product of each development step depends on the individual conceptual understanding, even with the same *thematician*, different teams may come up with dissimilar simulation models. This dissimilarity hinders reproducibility of results, which is one of the pillars of the *Scientific method*. One of the ways to ensure reproducibility is to perform verification from one step to another. Sargent [22] specifies two types of verification: the *specification verification* that takes place between the conceptual model and the simulation model specification and the *implementation verification* that takes place between the conceptual model specification and the simulation model. The complicating issue is that subject-matter experts, who form conceptual model by abstracting their understanding of the real-life problem entity, are not usually skilled in modelling and computer programming and cannot perform the verifications. They typically find the executable code obscure [4]. The definition of the model becomes 'hidden' in the simulation platform and cannot be perceived, validated or changed by the experts [4, 11]. Nevertheless, researchers urge the involvement of experts in the

creation of the model to ensure the validity of the simulation model specification and simulation model [4–6, 11, 21].

Consequently, we want to solve the following problem with this paper: How can we make ABMs more accessible to subject-matter experts to ensure verification and to enable validation of the simulation models? A possible approach favours building blocks that could enable non-programming experts to develop and modify their ABMs [10]. Continuing with this thought, we want to create a domain-specific language (DSL) and an associated tool allowing subject-matter experts to create and change the simulations models by changing their descriptions of conceptual models. Keeping this perspective in mind, we focus more on ease of use than efficiency in code generation.

This paper is structured as follows: Section 2 introduces ODD, NetLogo, and DSL and provides an overview of related work. Section 3.1 describes the process we followed to build each meta-model component. Section 4 presents our results and Section 5 discusses the quality of the outcome. Finally, Section 6 summarizes the paper. Throughout the paper, we illustrate the steps of the methodology with the Wolf Sheep Simple 5 model [25].

2 Background and Related Work

2.1 ODD

Many scholars in the Agent-Based Modelling community have adopted the Overview, Design concepts, and Details (ODD) protocol to further discussions among multidisciplinary researchers about their models. ODD emerged as an effort to "make model descriptions more understandable and complete, thereby making ABMs less subject to criticism for being irreproducible" as Grimm et al. state [7]. If we want to track the ODD description in the model development phases we would identify it as the simulation model specification or at the very least an intermediate step between the conceptual model and the simulation model specification. The protocol has seven thematic sections: "Purpose", "Entities, State Variables and Scales", "Process Overview and Scheduling", "Design Concepts", "Initialisation", "Input Data" and "Sub- models" [8]. Each section contains questions to guide modellers in the provision of related model details.

Figure 1, for example, shows the questions for the "Entities, State Variables and Scales" element of a model. The modeller provides the model definition by answering all the questions. The emerging document with the ODD specifications can be quite large, depending on the model it describes. The answers appear informally, and one can portray the protocol as a group of informal entities. The questions attempt to cover different perspectives of the conceptual model so that all researchers have eventually the same impression of what the model entails. Nowadays, many journals consider the ODD protocol as a prerequisite for the publication of an ABM model. Although this is a big step towards verification, validation and reproducibility of simulation models, the informal character of the answers allow ambiguities in the model description. Platforms such as the

Fig. 1. Informal ODD: Questions and Specification

ODD Element	Questions	Specification
Entity, State Variables, and Scales	What kind of entities are in the model? Do they represent managers, voters, landowners, firms or something else? By what state variables or attributes are these entities characterised? What are the temporal and spatial resolutions and extents of the model?	The agents represent sheep and wolves, and the environment is grassland that they inhabit. Both wolves and sheep have energy that they use to move around. On the other hand, the grass contains energy. The space represents a grassland and the time is not defined by the modeller, but, given the model dynamics, it should be within the lifetime of a sheep.

"ComSES Network OpenABM" mentioned in [12] enable the upload and sharing of ABMs in terms of executable code, ODD and other descriptions to promote model transparency and reuse and to move further towards a scientific handling of ABMs. There are more than 80 platforms that accommodate ABMs based on different programming languages [3]. SWARM covers a large range of models, MASON exhibits fast computational time and Repast is considered the best choice time-wise and modelling-wise [3, 19]. Unsurprisingly, all of them display shortcomings. For example, Repast is not well documented [19]. Our choice for a low-level language and simulation platform is NetLogo [26]. Uri Wilensky created NetLogo to facilitate the development of Agent-Based Modelling and Simulation. The platform has been widely used in the modelling community, and it is relatively simple for non-programmers to use [10], especially in the construction of ABMs [17]. Not only does NetLogo make it easier to develop a model, but it also provides an interface that facilitates simulations and reduces the amount of time needed to design them. In essence, a person with no modelling experience can explore a NetLogo model on the platform. Finally, NetLogo is an open source software.

2.2 DSLs and MPS

Domain-specific languages (DSLs) help efficient development and artefacts. However, it is not enough to have a DSL; one also needs an appropriate tool to work with the DSL. This is normally accomplished with a meta-tool creating a DSL tool out of a DSL description. Such meta-tool is called language workbench.

To build our tool, which we call ODD2ABM, we selected the Meta Programming System (MPS), a free platform for the creation of DSLs [13]. MPS provides projectional editing, which makes it easier to update a meta-model [9], in our case ODD2ABM. We create an ODD inspired editor so that the experience of importing the specifications resembles the experience of writing the ODD in a text file. A DSL description in MPS is structured in the aspects structure, edi-

tor, generator, and constraints. Finally, MPS provides tabular and diagrammatic notations in addition to plain text.

2.3 Related Work

Domain-specific languages together with Model Driven Development (MDD) [4] have often been used to solve problems similar to ours. DSLs aspire to provide the model definition in a high-level language so that experts can understand and modify the (domain-specific) model. MDD aims to automate the processes from the high-level description to the lower-level code. The processes of interest make use of languages with different abstraction levels and usually move from a high-level to a low-level language. The developer defines the transformations from one stage to the other. The end user inputs specifications in the high-level language and the DSL tool (semi)automatically generates the next stage artefact or the final code in the low-level language.

Some researchers have worked on the formalisation of certain aspects of ABMs, such as interactions [15]. Others have built meta-models that specialise within specific domains. The MAIA meta-model (Modelling Agent systems based on Institutional Analysis) covers Social Simulations with Institutional Analysis and semi-automatic code generation [6]. MDA4ABMS merges both DSL and MDD methodologies, but the user needs some modelling experience to handle the high-level language, and the tool does not automatically provide the low-level language artefact [4]. Also, the inclusion of UML (Unified Modelling Language) [18], which is applied in the methodology, has often been discarded by other researchers due to its lack of expressiveness [21]. Similarly, the easyABMS methodology includes UML and does not provide automatic code generation but takes into account all the modelling and simulation phases and can be used for general processes [5]. The meta-model introduced by Santos et al. [21] automatically generates code, and has been evaluated as very efficient; however, it only functions for a particular domain. Finally, adaptations of Multi-Agent methodologies to Agent-Based have been able to establish a common high-level formal language, but these do not include automatic code generation [11].

3 Methodology

This paper aims at creating a DSL and an associated tool for the construction and modification of ABMs. The central idea is that the user will input the conceptual model in the DSL and the tool will automatically generate the simulation model in NetLogo executable code. Apart from advantages related to the creation or modification time of conceptual and simulation models, the method provides build-in verification of the model. The main reason is that there is a deterministic relationship between input (formally described conceptual model) and output (executable code or simulation model). In essence, the verification is the tool itself. The goal is to start from simple models, so that we obtain a proof of concept for our idea, and then extend the work to more complex ABMs.

The tool should be easy to use, extensible, and allow automatic code generation from the model definitions produced by subject-matter experts. Our methodology integrates aspects of MDD and DSL. The DSL ensures user-friendliness and accommodates diverse models, while the MDD is used to provides the code generation into a lower level simulation language. Using MPS as the tool of choice for DSL development enables a focus onto the language description. MPS will generate a neat and efficient DSL tool directly and automatically from the language description.

As ODD already is a DSL for social models, a tool that transforms an ODD to NetLogo would solve our problem. However, ODD is not formal enough for a direct transformation. Still, we argue that instead of creating our DSL from scratch, using the methodologies proposed by [5, 11, 21], we can take advantage of the accumulated experience of researchers that the ODD incorporates and formalise it (i.e. we make the descriptions already in ODD more formal). Some of the advantages of using the protocol as a starting point include its pre-existing structure [14] and its inclusion of Agent-Based Modelling domain concerns that cover the need for the DSLs broadness and extensibility. Skipping the domain analysis and spending less energy to organise ODD, reduces the effort of the DSL development. Although Santos et al. [21] used the ODD protocol to refine the collection of concepts for the domain analysis of their case study, researchers have not yet taken full advantage of it to render ABMs more accessible.

We argue that NetLogo is a good starting point as the simulation language of our tool since it is well documented [1] and accommodates a variety of models (but not large scale ones). Since it is a higher-level language than general-purpose programming languages such as Java, it will be easier to design transformations from NetLogo to Java or similar programming languages in a later phase of our project.

Using this method, the two remaining challenges are: (1) a formalisation of the sections of ODD with important information for the simulation, and (2) a description of the transformation from ODD to NetLogo. The formalisation of ODD is closely related to the user friendliness of ODD2ABM. We want to make our DSL so accessible that it could be used by experts without any programming experience. This ease of use is intended to encourage and enable such users to construct and adapt AMBs without overly relying on computer scientists and coders. Moreover, we want to make our DSL capable of incorporating a broad range of models. Integrated perspectives support robust models such as [16]. In this example, Kuil et al. incorporated a fusion of social and ecological dynamics that managed to explain the decline of the Mayan civilisation [16]. Therefore, it is important to accommodate different types of models and not restrict users to a specific domain.

The two main challenges in the creation of a formal ODD are the repetition of information, and missing information. Missing information is information that is available in the NetLogo code, but that is not present in the ODD. For such information we have to find out whether the information can be generated from other existing information, or whether it must be included into the ODD.Repeated in-

formation could be handled by just ignoring the duplicated parts. However, it is important that the formal parts of ODD are reliable and if there is duplicated information, it has to be synchronized. In MPS this problem involves determining the primary place of the information and the creation of a reference to it at all the other uses.

One aspect of the tools friendliness is its capacity to automatically generate executable code, a task which has previously required programming skills. Using MPS, it is straightforward for non-programming experts to run their simulations. Although we have chosen NetLogo as target language for the code generation, there is still a lot of variability in the actual code to be produced. This again might influence the choice of concepts in the DSL, as we would prefer concepts that are easily implemented.

3.1 Meta-model elements

To create ODD2ABM in MPS, we needed to define the structure, constraints, editor, and generation rules of the DSL. ODD itself comes with an editor as shown in the Figure 1. ODD2ABM should use a similar editor reusing existing elements and adding new ones when necessary. It was not clear from the start which formal elements would be needed for ODD. To determine them, we used the following systematic procedure. The procedure organised the input in such a way as to ensure that the ODD user provides the data to cover the specifications. It resulted in a DSL description for structure and constraints. Finally, we created generation rules for automatic code generation.

3.2 Procedure for defining the Meta-model Structure

Collection of the ODD elements and questions We selected the ODD version from Grimm et al. [8], which includes seven ODD elements.

Selection of NetLogo models for concrete model instances and code Since this is the first version of ODD2ABM, we chose to start with simple models from the NetLogo library. The models we used are the Segregation Simple model [24], the Fire simple model [27], and the Wolf Sheep Simple 5 model [25]. For each of the models, we had an ODD description and NetLogo code. In parallel, we consulted the NetLogo dictionary [1] and a chapter specialised on Agent-Based modelling concepts for NetLogo [28]. The dictionary and the ABM concept overview ensured that the the simplicity of the first test models will not comprise the extensibility and capacity for variety of our meta-model.

Match of each element with the corresponding code For each element in the code of the selected models, we attempted to find matching information in the ODD description. For example, for the element “Entities, State variables, and Scales” and the entity “sheep” (see Figure1), we registered the code “breed [sheep a- sheep” and “sheep own [energy]”. Questions such as “What are the

temporal and spatial resolutions and extensions of the model?" are not semantically significant for the code. Using the final code, we distinguished the ODD elements that produce parts of the code from those that do not.

Identification of the parts of NetLogo code that cannot be extracted from the ODD specifications One example is the NetLogo entities, which can be turtles or patches. Turtles are moving agents, which can have capacities such as interacting with one another and recognising their location. We can think of patches as part of the grid on the simulation space. They cannot move, but they have properties that can change (such as colour). Patches interact with turtles but also with their neighbouring patches. Finally, NetLogo links are connection lines between the moving agents that indicate some sort of relationship. The distinction between the different entities is not visible in an ODD specification. We attempted to distinguish between low-level information that should not be included into ODD from higher-level information that should be included. The method to accomplish this was to formulate the information on the level of ODD. We excluded information not conceptually level and used a low-level way to produce it.

Creation of questions in the ODD language to accommodate code generation To follow up the previous example, we created questions on whether the entity is part of the environment or not. The questions were first captured in a flow diagram. Then they were incorporated in the editor description or editor structure. For example, there is a different place in the editor to define general entities and environmental entities. The same procedure applies for the rest of the model.The answers are the specifications of the meta-model, which enable the code generation. The specifications with no semantic significance for the code generation, such as the Purpose statement (see Figure 3 require an informal textual answer.

Grouping questions that reappear It is possible to locate the same questions in multiple positions of ODD. For example, to define the descriptive characteristics of the environment or the total population or the attribute of an entity we need to input the same type of specifications. There are two reasons for grouping this information: first to reduce the time for the DSL development and second to enhance the visual representation of the specifications by reducing the amount of information in them.

Extraction of the DSL structure from the diagram Figure 2 illustrates part of the concepts of the DSL that relate to the ODD element "Entities, State Variables and Scales". Each user input corresponds to either a DSL concept (for example the concept Entity) or to a DSL concept attribute (for example the attribute label of the concept Entity).

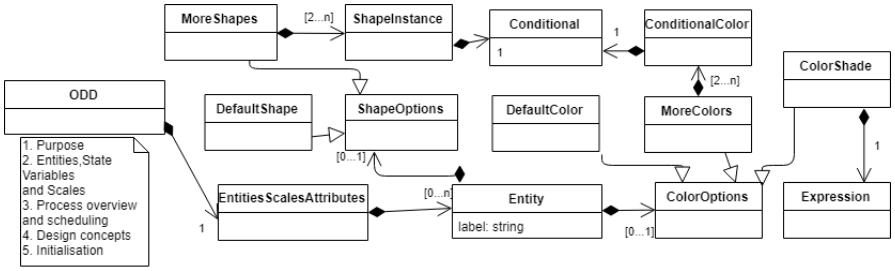

Fig. 2. UML class diagram of Entities, State Variables, and Scales

Registration of constraints that emerge from the structure Finally, we made sure that specifications are meaningful and did not violate common sense. For this, we extracted the informal conditions placed on the concepts and formalize them as MPS constraints. If we take the example of the entity sheep (Figure 3), we read that it contains the attribute "energy". The attribute is of type float. This will determine the next specification, which will be to define the range of values. If energy was of type string, then the editor would request a list of string values. In addition, the type of the variable and the specification "float" will constrain the input for other elements such as the initialisation. Finally, action is possible only for the defined attributes. The type of attribute selected in the specifications of "Entities, State Variables, and Scales" will constrain the value that the user imports in the Initialisation element. For example, the attribute "energy" was defined as "float". If the user were to assign a string value, the editor would not have accepted it. Overall, there are two types of constraints: those associated with the description in the specification and those associated with whether a specification is even available according to previous input.

3.3 Editor

The UML diagrams of the newly formalised ODD structure resemble the diagrams of the informal one in the sense that they expand from the seven elements. The identified questions of step five complement the initial ODD structure. The editor for ODD is providing textual and tabular syntax for the structural elements of ODD. The original look and feel of ODD is kept by putting the elements into text as much as possible and keeping the concepts at the same language level. The editor of MPS provides state-of-the-art content already from the start. Some more advanced features can be added manually. In particular, for the case of ODD it is important that the required input is very specific, except for answers with no semantic significance. For example, we can see in Figure 3, that the editor description is "Color is defined for the entity". The initial text is "Color ¡Press Alt and Enter to choose to include or not include color¿ defined for the entity". When the user presses Alt and Enter the only choices are "is" and "is not". The MPS editor provides auto-complete to help users identify possible

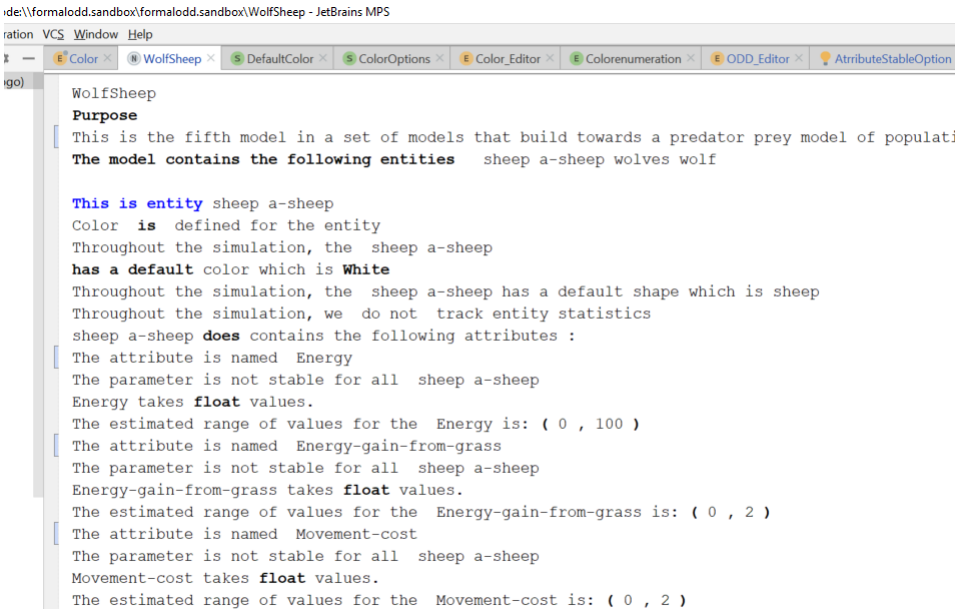

Fig. 3. Formalised ODD in MPS

continuations and lists where we want to define the possible answers, as well as static checks on the fly in order to avoid wrong inputs. For more complex inputs, the editor description indicates and checks the right way to configure the text. Because of the projectional editor poperty of MPS, it is not possible to see questions that are not enabled in the current model. For example, in the previous example, when the user selects "is not" then the part where the user specifies the color method and the specific color choice, currently visible in Figure 3, are not shown. The editor appearance is derived from the questions, which is similar to the corresponding concepts (see Figure 2). Overall, the elements are connected and the full UML diagram shows all the connections.

3.4 Executable Code Generation

Normally, code generation follows the flow of information as given in the ODD structure. Still, depending on the place in the generated code, it might be the case that more information is collected from different parts of the specification. For example, even though we choose to specify whether an entity has a color in the Entities State Variables and Scales element, MPS uses these specifications to generate code in the Initialisation part.

Code generation requires the specification to be statically correct, i.e. all constraints should be satisfied. This is checked already in the editor and signalled to the user.Code generation is normally disabled as long as there are errors in the

specification. Vice versa, utmost care was given to make sure that the code generation is successful whenever the specification is statically correct. For example, if we look at Figure **??**, generating the code "sheep own [energy]" requires the user to assign the attribute "energy" to the entity "sheep". If no entity has been defined, it will not be possible to define an attribute for it. Moreover, the label of the attribute has to be defined in order to enable generation of correct code. Sometimes code is only generated under specific circumstances, which explains why some rows in the "Generated Code" column are empty.

We can identify two categories of code in NetLogo: the code for the model definition and the code for the simulation setup. In our DSL, we do not differentiate between the two types. To the best of our knowledge, all models in NetLogo contain the Setup and the Go buttons in the NetLogo interface. Using these buttons, the modeller can restore the entities to the initial condition and start/stop the simulation. Even if the particular naming of the buttons ("Setup" and "Go") is not mandatory, the code does not run without the existence and pressing of a button in the user interface. All procedures are initiated when called directly or indirectly from one NetLogo button. Therefore, we choose the generation "Setup" and "Go" buttons by default in the code generation from the DSL.

4 Solution

We performed the eight-step procedure outlined in Section 3.2 for the four ODD elements: "Purpose", "Entity, State Variables, and Scales", "Process Overview and Scheduling", and "Initialisation". For each one, we created a Diagram that summarised the meta-model specifications and extracted the UML diagram. We verified the tool for the first 3 elements using the 3 NetLogo models (the Wolf-Sheep Simple 5, Segregation and Fire model). Figure 2 shows part of the structure for the Entities element and Figure 3 shows the editor from which the code "breed [sheep a- sheep" and "sheep own [energy]" was generated. The editor text serves as the ODD document.

5 Evaluation

5.1 Expressivity and Extension

We created our DSL based on relatively simple models and so cannot guarantee that ODD2ABM covers the range of model specifications that are needed for social simulation. However, the concepts are carefully chosen to cover a broad range of application such that there is at least a major range of possible specifications. In the future, we will validate the DSL with more complex ABMs and introduce more specifications. For example, we plan to further develop several aspects of the tool such as how it deals with interactions and relationships among agents marked in NetLogo with links, and add the possibility of importing data from files external to the platform. The extensibility of our DSL is ensured by MPS and the conceptual framework we adopted.

5.2 ODD and Experts

The ODD protocol gave us the opportunity to enrich it as a DSL with some modifications without losing its accessibility for users with limited programming skills. The question remains whether the level of the language is high enough for thematicians to engage with it. Part of the concern lies in the fact that the original ODD targets modellers. We all use models (in a general sense) in our everyday lives. However, the concepts employed in the Agent-Based Modelling community, such as entities and attributes, may not be intuitively clear to all users. Therefore, even if the tool is very effective for modellers, experts not familiar with the ODD language may face difficulties in its implementation. In a next stage of development we will evaluate the usefulness of ODD2ABM.

6 Summary and Future Steps

The ODD2ABM tool described in this paper serves as a proof of concept for a methodology that incorporates DSL and MDD, uses the MPS platform, enables experts to create, modify their ABMs and provides a new way to ensure reproducibility of results. During the construction of this tool, we were careful to ensure its user-friendliness and extensibility. We selected the ODD protocol as the basis for our DSL and NetLogo as our low-level language. The resulting DSL is original in its capabilities and properties as it accommodates a large range of modelling themes and enables automatic executable code generation. We plan to broaden ODD2ABM so that it allows more freedom in model creation. A next step would be to survey experts from different disciplines to discover whether our formalisation of the ODD protocol needs to be abstracted further in a process such as the one defined in [20].

References

1. Netlogo dictionary, https://ccl.northwestern.edu/netlogo/docs/dictionary.html
2. Abar, S., Theodoropoulos, G.K., Lemarinier, P., O'Hare, G.M.P.: Agent based modelling and simulation tools: A review of the state-of-art software. Computer Science Review 24, 13–33 (2017)
3. Abar, S., Theodoropoulos, G.K., Lemarinier, P., OHare, G.M.: Agent based modelling and simulation tools: A review of the state-of-art software. Computer Science Review 24, 13 – 33 (2017), http://www.sciencedirect.com/science/article/pii/S1574013716301198
4. Garro, A., Parisi, F., Russo, W.: A Process Based on the Model-Driven Architecture to Enable the Definition of Platform-Independent Simulation Models, pp. 113–129. Springer Berlin Heidelberg, Berlin, Heidelberg (2013), https://doi.org/10.1007/978-3-642-34336-0_8
5. Garro, A., Russo, W.: Easyabms: A domain-expert oriented methodology for agent-based modeling and simulation. Simulation Modelling Practice and Theory 18(10), 1453 – 1467 (2010), http://www.sciencedirect.com/science/article/pii/S1569190X10000717, simulation-based Design and Evaluation of Multi-Agent Systems

6. Ghorbani, A., Bots, P., Dignum, V., Dijkema, G.: Maia: a framework for developing agent-based social simulations. Journal of Artificial Societies and Social Simulation 16(2), 9 (2013), http://jasss.soc.surrey.ac.uk/16/2/9.html
7. Grimm, V., Berger, U., DeAngelis, D.L., Polhill, J.G., Giske, J., Railsback, S.F.: The odd protocol: A review and first update. Ecological Modelling 221(23), 2760 – 2768 (2010), http://www.sciencedirect.com/science/article/pii/S030438001000414X
8. Grimm, V., Polhill, G., Touza, J.: Documenting Social Simulation Models: The ODD Protocol as a Standard, pp. 117–133. Springer Berlin Heidelberg, Berlin, Heidelberg (2013), https://doi.org/10.1007/978-3-540-93813-2_7
9. Guttormsen, S., Prinz, A., Gjøsæter, T.: Consistent projectional text editors. In: MODELSWARD. pp. 515–522 (2017)
10. Hamill, L.: Agent-based modelling: The next 15 years. Journal of Artificial Societies and Social Simulation 13(4), 11 (2010), http://jasss.soc.surrey.ac.uk/13/4/7.html
11. Hassan, S., Fuentes-Fernández, R., Galán, J.M., López-Paredes, A., Pavón, J.: Reducing the modeling gap: On the use of metamodels in agent-based simulation. In: 6th conference of the european social simulation association (ESSA 2009). pp. 1–13 (2009)
12. Janssen, M.A., Alessa, L.N., Barton, M., Bergin, S., Lee, A.: Towards a community framework for agent-based modelling. Journal of Artificial Societies and Social Simulation 11(2), 6 (2008), http://jasss.soc.surrey.ac.uk/11/2/6.html
13. JetBrains: MPS Meta Programming System, https://www.jetbrains.com/mps/
14. Klügl, F., Davidsson, P.: Amason: Abstract meta-model for agent-based simulation. In: Klusch, M., Thimm, M., Paprzycki, M. (eds.) Multiagent System Technologies. pp. 101–114. Springer Berlin Heidelberg, Berlin, Heidelberg (2013)
15. Kubera, Y., Mathieu, P., Picault, S.: Interaction-oriented agent simulations: From theory to implementation. In: Proceedings of the 2008 Conference on ECAI 2008: 18th European Conference on Artificial Intelligence. pp. 383–387. IOS Press, Amsterdam, The Netherlands, The Netherlands (2008), http://dl.acm.org/citation.cfm?id=1567281.1567367
16. Kuil, L., Carr, G., Viglione, A., Prskawetz, A., Blschl, G.: Conceptualizing socio-hydrological drought processes: The case of the maya collapse. Water resources research 52(8), 6222–6242 (2016)
17. Lytinen, S.L., Railsback, S.F.: The evolution of agent-based simulation platforms: A review of netlogo 5.0 and relogo. In: Proceedings of the fourth international symposium on agent-based modeling and simulation (21st European Meeting on Cybernetics and Systems Research [EMCSR 2012 (2012)
18. OMG: Unified Modeling Language: Infrastructure version 2.4.1 (OMG Document formal/2011-08-05). OMG Document, Published by Object Management Group, http://www.omg.org (August 2011)
19. Railsback, S.F., Lytinen, S.L., Jackson, S.K.: Agent-based simulation platforms: Review and development recommendations. SIMULATION 82(9), 609–623 (2006), https://doi.org/10.1177/0037549706073695
20. Santos, F., Nunes, I., Bazzan, A.L.C.: Supporting the development of agent-based simulations: A dsl for environment modeling. In: 2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC). vol. 1, pp. 170–179 (July 2017)
21. Santos, F., Nunes, I., Bazzan, A.L.: Model-driven agent-based simulation development: A modeling language and empirical evaluation in the adaptive traffic signal control domain. Simulation Modelling Practice and Theory 83, 162 – 187

(2018), http://www.sciencedirect.com/science/article/pii/S1569190X17301673, agent-based Modelling and Simulation
22. Sargent, R.G.: Verification and validation of simulation models. In: Proceedings of the 2010 Winter Simulation Conference. pp. 166–183 (Dec 2010)
23. Wildman, W.J., Fishwick, P.A., Shults, F.L.: Teaching at the intersection of simulation and the humanities. In: 2017 Winter Simulation Conference, WSC 2017, Las Vegas, NV, USA, December 3-6, 2017. pp. 4162–4174 (2017), https://doi.org/10.1109/WSC.2017.8248136
24. Wilensky, U.: Netlogo seggregation model. Report, Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL (1997), http://ccl.northwestern.edu/netlogo/models/Fire
25. Wilensky, U.: Netlogo wolf sheep predation model. Report, Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL (1997), http://ccl.northwestern.edu/netlogo/models/WolfSheepPredation
26. Wilensky, U.: Netlogo. Report, Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL (1999), http://ccl.northwestern.edu/netlogo/
27. Wilensky, U.: Netlogo fire model. Report, Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL (2006), http://ccl.northwestern.edu/netlogo/models/Fire
28. Wilensky, U., Rand, W.: The Components of Agent-Based Modeling, pp. 203–282. The MIT Press, Cambridge, Massachusetts ;, 1st edn. (2015)
29. Wurzer, G., Kowarik, K., Reschreiter, H.: Agent-based modeling and simulation in archaeology. Springer (2015)