Transparent and Accessible ABMs with FODD: Automatic Code from Formal ODD

Themis Dimitra Xanthopoulou
Umeå University
Umeå, Sweden
themis.xanthopoulou@umu.se

Andreas Prinz
University of Agder
Grimstad, Norway
andreas.prinz@uia.no

Haakon Bøthun Lunde
University of Agder
Grimstad, Norway
haakbl99@gmail.com

Ivan Puga-Gonzalez
NORCE
Kristiansand, Norway
ivpu@norceresearch.no

F. LeRon Shults
University of Agder
Kristiansand, Norway
NORCE
Kristiansand, Norway
lesh@norceresearch.no

ABSTRACT

Agent-based models (ABMs) face a persistent transparency challenge: open-source code rarely guarantees understanding for non-programmers, while readable documentation may not faithfully reflect the implementation. This documentation–code gap undermines verification, peer review, and interdisciplinary collaboration—problems that are acute for multi-agent models of artificial societies. We introduce FODD, which establishes a verified link between model documentation and executable code by extending the widely used Overview, Design concepts, and Details (ODD) protocol with formal specifications that automatically generate NetLogo code, preserving a structure familiar to domain experts. We describe the FODD language as implemented in JetBrains MPS, covering its structure, editor, constraint mechanisms that enforce type safety and completeness, and transformation rules for deterministic code generation. Our proof-of-concept tool separates formal, informal, and derived content, derives parts of the design documentation from behavioral and initialization specifications to reduce redundancy, and explicitly links entities, attributes, and procedures to reduce conceptual ambiguity. We tested feasibility with six diverse models and evaluated the tool along executability, expressivity, and user-friendliness. The tool generates runnable models with aligned documentation, but the behavior specification language still needs higher-level abstractions to be intuitive for domain experts. FODD has the potential to make model development more efficient by reducing manual coding stages and verification effort. At its current proof-of-concept stage, FODD primarily supports experienced modelers while laying the groundwork for broader domain-expert participation in model inspection, discussion, and validation. In this sense, it advances transparency, reproducibility, and participatory modeling in computational social science.

Proc. of the 25th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2026), C. Amato, L. Dennis, V. Mascardi, J. Thangarajah (eds.), May 25 – 29, 2026, Paphos, Cyprus. © 2026 International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org). https://doi.org/10.65109/XDBV4230

KEYWORDS

Agent-Based Modeling; Multi-Agent Systems; Automatic Code Generation; Model Verification; Domain-Specific Languages; Model Documentation; Social Simulation; Executable Specifications; Transparency; ODD; FODD

ACM Reference Format:
Themis Dimitra Xanthopoulou, Andreas Prinz, Haakon Bøthun Lunde, Ivan Puga-Gonzalez, and F. LeRon Shults. 2026. Transparent and Accessible ABMs with FODD: Automatic Code from Formal ODD. In *Proc. of the 25th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2026), Paphos, Cyprus, May 25 – 29, 2026*, IFAAMAS, 9 pages. https://doi.org/10.65109/XDBV4230

1 MOTIVATION

In 2020, the question of model transparency spurred the interest of the Social Simulation community when policymakers based the measures for COVID-19 on a simulation model with undocumented code, eliciting calls for action by researchers [26]. Using models for prediction is extremely challenging (if not impossible) [10], and models with unverified parts used for prediction in a crisis environment could actually cost lives. Growing awareness of the risks models can pose in the hands of people who do not understand them puts even more pressure on increased transparency.

The demand for transparency has only grown since then, with recent contributions stressing the importance of reproducibility and documentation. For example, recent work highlights how failures in code and data availability undermine trust in simulation studies [16], while others provide concrete recommendations for communicating ABMs more clearly to stakeholders and ensuring their reuse [3]. In parallel, scholars have argued that weak documentation is a critical barrier to policy engagement, especially in light of new regulatory frameworks such as the EU AI Act, which further elevates the need for transparent and evaluable models [5].

At its core, transparency ensures access to the model and is therefore required for procedures that assess model quality. Views on model quality differ and reflect factors such as philosophical assumptions about reality [2, 13]. Nonetheless, model quality is commonly linked to verification and validation and to the collaborative processes through which models are built [2]. Across the model

lifecycle, access is repeatedly required: during verification and validation (to compare expert or stakeholder expectations with model contents and outputs), during peer review (to scrutinize claims and implementation), and after publication (to enable assessment, reuse, and replication). However, in practice, access is not guaranteed: source code is often inaccessible to non-programmers, and natural-language documentation cannot be mechanically checked against the implementation and may drift as models evolve, leading to a documentation–code gap.

Even when we have model access, it is not enough to demystify the model contents. To understand this, consider model code. One logical step toward model transparency is to make code open source. Although this enables replication, it does not necessarily help reviewers or domain experts understand the model. Domain experts are not expected to have programming skills. Moreover, even when one can read the code, it does not necessarily result in a correct overview of the model. Even modelers with deep programming expertise can make implementation errors. If such experts cannot reliably detect these errors while reviewing their code, then readers may miss key details or misunderstand how the code behaves during simulations. Finally, it is impractical to try to understand a model solely from its code [24].

To improve model readability and make modeling choices explicit, modelers use a range of documentation techniques. Among these, the ODD protocol is by far the most widely used in the social simulation community [14]. Modelers answer the protocol's guiding questions to describe model structure and behavior, with questions designed to surface key modeling decisions and provide a basis for discussing and critiquing a model's contents. However, ODD is not a verified document. It has no formal link to the implementation and can therefore diverge from the code and potentially misrepresent it [24]. The 2020 update acknowledges this ambiguity [15] and suggests reducing it by linking ODD descriptions to specific code locations. More broadly, maintaining verified alignment between agent specifications and implementations is a recurring challenge across multi-agent systems. Accordingly, a variety of documentation protocols exist across paradigms, from ODD extensions such as ODD+D and ODD+2D and ABM reporting standards such as RAT-RS and KIA to more general frameworks such as TRACE, ODMAP, OPE, and PERFICT (see [21], for a review).

We argue that a formal link between ODD-style documentation and model code can improve transparency and readability. To engineer this link, we adopt a Domain-Specific Modeling (DSM) approach [20] by implementing FODD as a domain-specific language in JetBrains Meta Programming System (MPS) [19], a language workbench for defining languages together with editors, constraints, and generators. Our scope focuses on social simulation agent-based models, where the world is conceptualized using an environment, agents or entities, behaviors, and properties [1].[1] This paper contributes an operational Formal ODD toolchain (FODD) and reports its design and use in practice: (1) the FODD language and editor in JetBrains MPS, (2) static checks and completeness mechanisms offered to users of FODD, (3) deterministic transformation rules implemented in MPS that generate NetLogo code from FODD specifications, and (4) initial feasibility evidence across six diverse models. Our earlier work describes the methodology for building Formal ODD [30], but in this paper, we focus on the operational artifact and how these components work together in practice.

Section 2 introduces FODD, explaining how it creates a formal link between ODD documentation and executable NetLogo code and outlining the resulting benefits for transparency and development workflow. Section 3 describes the FODD language and tool support, covering its structure, editor interface, constraint mechanisms, and code generation rules. Section 4 details how FODD implements and extends ODD protocol elements, including derived sections, entity–procedure links, behavior specifications, and experiment design formalization. Section 5 reports initial feasibility results across six models for our proof of concept in terms of executability, expressivity, and user-friendliness, and outlines our development roadmap. Finally, Sections 6 and 7 discuss limitations and summarize the paper's contributions.

2 SOLUTION

ODD guidelines, which resemble conceptual model descriptions, ask modelers to justify design choices and describe key model aspects such as behaviors [23], but the result depends on the author's rigor and can still support multiple implementations. There is also no practical way to check that an ODD matches the code beyond tedious manual comparison, even though domain experts, stakeholders, policy makers, and reviewers rely on these descriptions when assessing models and using their results.

A Formalised ODD (FODD) narrows the documentation-code gap because selected ODD sections become typed, checkable executable specifications that preserve the ODD structure, while keeping other sections informal or derived (see Table 1). The transformation rules deterministically map the FODD specification to executable NetLogo code, so changes to the specification update the runnable model and keep documentation aligned with the executable model. We formalize ODD rather than another template because it is widely used, actively maintained, and validated in practice. In addition, the FODD specification remains at a higher level of abstraction than the generated code, closer to the conceptual level of ODD, which makes it accessible and transparent to stakeholders without programming expertise. This should not be confused with natural language, since the formal parts of FODD have explicit syntax and semantics enforced by static checks and deterministic transformation rules. FODD is designed to keep the core model specification backend-independent. In the current prototype, however, some experiment and interface constructs remain NetLogo-specific, so backend independence presently applies mainly to the conceptual layer of the specification. Supporting other platforms would require additional generator mappings and the generalization of target-specific runtime and interface conventions.

2.1 User Workflow

The workflow is minimally presented in Figure 1. In the beginning, the user specifies the model in the FODD editor in a process analogous to completing an ODD description, but here the specification and documentation are produced simultaneously, avoiding intermediate pseudocode and code description stages [12]. The editor

[1] In this paper, we use both terms *agents* and *entities*.

guides the user toward complete specifications (see Section 3 and Figure 2). Once the specification is complete, the user initiates code generation with a single click, producing NetLogo code that can be imported into the simulation platform, which we selected for reasons described in [30]. The user then runs experiments in NetLogo and proceeds with data analysis.

Figure 1: FODD workflow from specification to generated NetLogo code

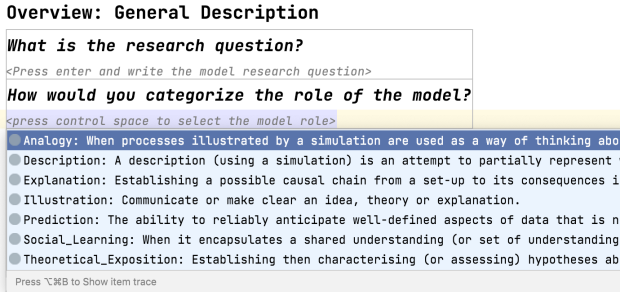

Figure 2: Overview Section in the Tool. Drop-down menu to select the role of the model.

2.2 Benefits and Optimization

FODD can reduce model development time by eliminating the approximation and coding stages as described in [12]. A single specification is checked by constraints and deterministically transformed into code, simplifying updates and reducing implementation errors. Documentation effort drops because some ODD sections are derived automatically (Table 1). Since the specification mirrors ODD and remains aligned with the generated code, it supports verification [13] and participatory modeling [4] without requiring domain experts to read source code. The current notation still targets modelers more than domain experts (Section 5).

3 FODD LANGUAGE

We designed a DSM tool that implements a domain language and defines transformations to a target programming language. We chose NetLogo [29] as our initial target because NetLogo is a commonly used platform for ABMs with a friendly interface to run simulations for non-programmers. We implemented the formalized ODD language in JetBrains MPS [19], a language workbench that supports multi-language tool building via metamodeling. In this approach, a language is defined through four parts [6]: structure (abstract syntax), editor (concrete syntax), constraints (static semantics), and generator (dynamic semantics). Our editor mirrors the ODD document structure (Table 1). We refined these four parts iteratively using agile principles [22] and tests on NetLogo Library models. The next sections describe each part; methodological details are reported in [30]. FODD denotes the formal domain language and ODD-structured model description. ODD2NetLogo is our current JetBrains MPS-based implementation and generator targeting NetLogo.

3.1 Structure

The structure defines the available concepts as a hierarchy. The root concept is ODD, followed by tool concepts that match ODD elements (Table 1, Column 2) and refine into lower-level concepts such as “entities”, “user-defined attributes”, and “sliders”. Concepts in MPS can have attributes, for example “sliders” include “min-value”, “max-value”, and “increment”. The main reusable concepts are *Expression* and *Activity*. One use of the concept “Expression” is to assign values to variables (numbers, booleans, or arrays) via constant values, sliders, or distributions, and it also builds string values for conditions and grouping of entities. UML diagrams of the structure can be found on GitHub [8].

3.2 Editor

The editor is the interface of the tool and the main focus of this paper. The editor description requests the user to insert certain information about the model in the ODD style. When the user signals the existence of an entity or procedure in the model, the editor creates an initial description that requests the information needed to completely define it. The formality of the tool requires the user to give more details about the model than the ODD protocol. However, the user of the tool is not obliged to know how to formally structure the information. To improve usability, the editor provides context-sensitive drop-down menus for many inputs (Figure 2) and template-based guidance embedded in the editor (Figure 3). To ease the input of specifications, we designed much of the editor to present template-based information (see for instance Figure 3) with fill-in sections rather than questions to be answered. With this design, we also aim to reduce the size of the specification.

Overall, the specification supports communication about the model and enables automated checking for technical errors. Because generation is deterministic, the same specification always produces the same NetLogo output, supporting replicability. We do not offer a new modeling methodology and we cannot change the potential of the output model. We only provide an enhanced means of transparency, verification, replication and communication.

3.3 Constraints

The main role of constraints is to ensure that model specifications are meaningful and that user input is valid. In FODD, this is enforced through static checks using specific rules, utilizing MPS’s constraint, behavior, and typesystem aspects.

The constraint and behavior aspects determine which inputs are available from drop-down menus (such as the one shown in Figure 2) and provide contextual instructions. These aspects are primarily employed to impose constraints on concepts with narrow use cases, such as the “All” concept which is only available when selecting a set using the “SelectN” concept, or to determine the scope of variables. Concepts that are assigned types—numbers have the numerical type, while user-defined variables have logic that determines their type. An example of a static type-check is verifying an assigned value against the defined variable type. A

Table 1: Matching ODD elements to Tool Sections and Description of contents.

ODD element	*Section in the Tool*	*Description of Section*
Overview		
Purpose and patterns	Purpose	Informal
Entities, state variables, and scales	Entities, state variables, and scales	Formal
Process overview and scheduling	Process overview and scheduling	Formal
Design Concepts		
Basic Principles	Rationales	Informal
-	Use of entities and attributes in Procedures	Derived
Emergence	Emergence	Informal
Adaptation	-	(Derived)
Objectives	-	(Derived)
Learning	-	(Derived)
Prediction	-	(Informal)
Sensing	-	(Derived)
Interaction	Interaction	Derived and Informal
Stochasticity	Stochasticity	Derived and Informal
Collectives	Collectives	Derived
Observation	-	(Derived)
Details		
Initialization	Manual Experiments	Formal
Input Data	-	-
Submodels	Process overview and scheduling	Formal
-	Experiments	Formal and Informal

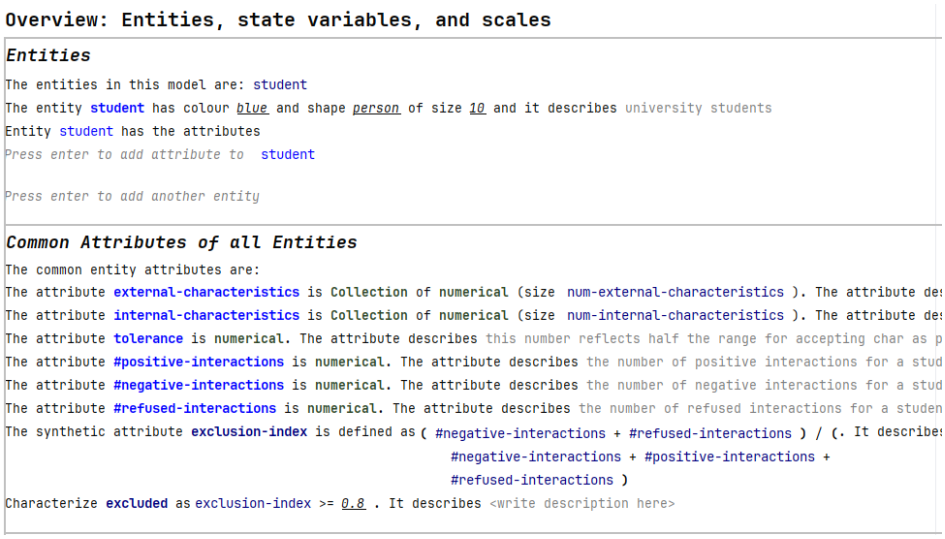

Figure 3: Section Entities, State Variables, and Scales in the Tool.

divergence from the defined type, such as passing a "wolf" entity to an action defined for "sheep", or passing an entity to an action defined for an environment entity, will cause an error. Relevant concepts implement type-checks to ensure that relationships are meaningful. For instance, the "IncrementAttribute" concept only has defined behavior for numbers, so type-checks are performed on both the value to be incremented and the value to increment by to test if both are of the number type. User input that does not adhere to the typesystem rules returns errors in the common format of a red underline or an error message.

3.4 Generator

The transformation of input to code is handled by the generator aspect. The generation rules are an inherent part of the tool and are documented based on the editor elements. The rules consist of parametrized templates that correspond to the FODD language

concepts as well as rules that determine which template is used and how they are populated with user input. They belong to the generator aspect of MPS and can be reviewed and updated. Sometimes, the generation rule uses input from different sections of the editor description. The generation rules imply that we have made choices when it comes to the style of the code and the semantics. An example of the code style is the selection of the programming paradigm to be generated from our tool within the limitations of NetLogo. Another example is how we manage the sequence of procedures. The overarching architecture of the output is defined in the root-template. This template is populated with user input and written to a netlogo-file. Such decisions make the editor less technical and allow the use of the tool by interested parties who are not experienced in programming. From this short overview, it becomes apparent that a language extension requires at least one new concept with a new editor description and a new generation rule.

Syntactically incomplete FODD specifications are detected mainly through structural completeness rules in the MPS metamodel. If a field that is required by the structure (e.g. a relationship whose cardinality is not optional) is left empty, the editor flags it as an error and points to the exact location of the "illegally" empty field. We do not guarantee correct code generation for incomplete specifications.

The projectional editor also prevents another common form of incompleteness: users cannot reference variables that do not exist or that are out of scope, so "undefined reference" specifications cannot be constructed. The main limitation is that this only catches incompleteness that violates structural or typing requirements: if a user's partially finished specification is still semantically valid under the current constraints, it may generate code without errors even if the model is not "complete" in the user's intended sense.

4 FORMALITY AND FODD IMPLEMENTATION FEATURES

We distinguish three levels of formality for user specifications: formal specifications, informal specifications, and derived. Table 1 shows how FODD sections match ODD protocol elements, with the third column indicating formality levels.

FODD addresses a fundamental challenge in agent-based modeling: conceptual ambiguity. The sloppy use of concepts can make computer models misleading, as human brains naturally confuse or mix concepts. When we name entities in our models, readers' associations with those names may lead them to expect connections we did not intend. This is especially problematic for ambiguous concepts such as bullying [31]. FODD addresses this challenge by explicitly connecting entity definitions to their attributes and procedures (Section 4.2), and by using concrete actions to specify behavior (Section 4.3), thereby exposing how concepts are actually implemented in the model.

In the following sections, we explain how these different aspects work. The formal specifications establish a verification link between documentation and code, ensuring that each time the user provides the same specifications, the same model will be generated by the tool. This contrasts with ODD descriptions, which can prescribe multiple potential model implementations depending on the amount of information provided and how semantics are interpreted by individuals. Unlike code alone, the FODD description provides contextual information that helps situate the model and specify the application context, as well as information about modeling decisions that helps readers evaluate its quality. Additionally, FODD formalizes experiment design, which aids in exposing how the model purpose is explored in practice (see more in Section 4.4).

Throughout these sections, we illustrate key features using the model "MARG" [32]. In this model, university students choose whether or not to interact with random partners based on their preferences in terms of characteristics. Their selections lead to a perceived inclusion or exclusion, which varies from student to student. More information on the model can be found in [32]. The model code and documentation are available in [27], which includes the informal ODD protocol of the model and the model code. [2]

4.1 Design and Information Derived from Formal Sections

The ODD element *Design Concepts* is to expose why modelers make certain modeling decisions and how they connect to the real world. As suggested in one of the ODD updates [23], modeling decisions could rely on established theories, real-life observations, ad hoc rules or a combination of the above. ODD proposes the following Design Concepts: *Basic Principles*, *Emergence*, *Adaptation*, *Objectives*, *Learning*, *Prediction*, *Sensing*, *Interaction*, *Stochasticity*, *Collectives*, and *Observation* [15].

We use three ways to handle design concepts in our tool: the first is to collect information from the formal specifications, and the second is to request the user to write information informally as they would in the ODD protocol. The third way is a mix of the first two. We show the difference by looking at the Design concept of Stochasticity for the model "MARG". According to [15], under *Stochasticity*, one should be able to record "the processes that are modeled as stochastic" and the Rationale behind them. When the user sets up the procedures in **Process Overview and Scheduling** and **Initialisation**, they register whether randomness is invoked. Since this information is already stored in the formal specification, ODD2NetLogo collects it and presents it in the section **Stochasticity**, see Figure 4. However, information about why randomness is used is not available in the formal parts. Therefore, the user must add it as informal text to fulfill the Rationale requirement in **Stochasticity**. As a result, **Stochasticity** contains both **Derived and Informal** [3] type of Sections as indicated in Table 1. Derived sections ensure that the user does not manually register information already available in other specifications. By doing that, we limit the user's potential for errors when writing data in the ODD. Finally, this method matches the requirement of the ODD protocol [15] and eases the documentation of the **Design Concepts**.

[2] Note that the images correspond to the version of our implementation at the time of the paper writing.

[3] Currently the Information sections do not show in Figure 4 but Figure 5 shows informal sections for other design concepts (look at "has rationale based on").

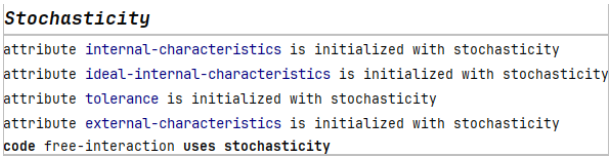

Figure 4: Subsection Stochasticity of Section Design Concepts in the tool.

4.2 Entity Essence via Attributes and Procedures

When we name the entities in our models, readers' associations with those names may lead them to expect connections we did not intend. The ODD protocol has devoted a whole section to clarify the intended meaning behind the words. A dedicated modeler will cover the aspects that are coded. However, it is more challenging to convey what is not modeled. Even the most diligent modeler is unaware of all the associations the brain makes with each name. Even if they were, there is no reliable way to cover the associations other people make regarding the same concept. This is especially true for ambiguous concepts such as bullying [31].

Apart from the benefits of a formal connection between description and code, we address the ambiguity of entity concepts by connecting different sections in the tool. We argue that the essence of entities lies in the attributes and procedures of the model and that we must explicitly link them. We accomplish this by presenting them together. Attributes specify which aspect of the entity constitutes the model focus. For instance, the entity "student" in the model "MARG" includes attributes such as "tolerance", "internal characteristics", and "negative interactions". By contrast, a very different idea of the entity student is used in the model in [28]. In the latter, the entity owns the attributes: "score", "preference-list", and "choice".

We can build many models with the same attributes. To further clarify the narrative behind the model, we expose the procedures related to entities and attributes in the formal version of ODD. Figure 3 shows a part of the element *Entities State Variables and Scales* for the model "MARG". First, the user defines model parameters—global attributes that influence the model. Then they must define the entities, the "students" for the model "MARG", and state their unique attributes. In the next section, the user is requested to input the shared attributes of all entities. Then, the user will describe the networks among entities. Figure 5 shows an overview of all procedures the entity "student" is connected to, offering enhanced clarity on the meaning of "student". Finally, the user registers synthetic values. Synthetic values are derived from other values using some expression and indicate properties that can be computed. For instance, in the model "MARG", based on the type of interactions (agent attributes) an agent has, the attribute "exclusion index" is calculated and then based on thresholds, the agent is assigned the boolean value of "included" or "excluded".

4.3 Using Concrete Actions to Describe Procedures

Two gaps can lead to unintended interpretations: structural incompleteness and ambiguity. When required structural fields are missing or types are inconsistent, the editor flags errors and code generation is blocked. Ambiguity is addressed by deconstructing a procedure name with other procedures. Figure 6 shows the definition of the procedure **forced-interaction** for the model "MARG". The definition of the procedure **forced-interaction** uses two other procedures: **learn-about** and **evaluate-interaction**. Then each procedure is further analyzed with the use of activities. The tool supports several types of activities such as **moving**, **creation of entity**, and **death of entity**. The activities are also selected using templates. The activities are of lower granularity than procedures. Thus, they can relate to real-life observations and help clarify procedures. After the definition of procedures, we need to schedule them. For the model "MARG", the scheduling uses the two procedures **forced-interaction** and **free-interaction** (shown in Figure 7).

4.4 The Purpose of the Model and the Experiments

We create new models because we want to get something out of them. There are at least 16 reasons to model [11], which can be further categorized into 7 general modeling purposes [9]. For the ODD protocol [14], the identification of the model purpose is the first step. The purpose defines the lens through which we model. As such, the purpose directs the entities we choose and their attributes, the processes we model, the experiment design, and the data collection. All of this is, of course, restricted by the computational power available.

It sometimes happens that we design experiments outside of the model's original purpose in order to explore it further. Additional experiments are welcome when we understand the limitations and opportunities that the model purpose has enforced on the model. For instance, the model "MARG" was designed to guide ethical reflection on issues of bullying, defined as intensified marginalization [32]. We can only offer abstract insights if we use the same model to test interventions. However, if we later assess that the model is fit to accommodate a new purpose, we can use it to explore other issues. This assessment is crucial to avoid what [25] calls the Portability trap—the tendency to reuse software artifacts in contexts they were not designed for without adequately considering how the original design constraints limit their applicability. In agent-based modeling, this means explicitly documenting both the original purpose and the implications of any purpose extensions.

In the FODD, the user needs to answer some questions regarding the modeling purpose, such as **What is the research question?** and proceed to select which purpose category expresses best the model purpose (see Figure 2). For this part, as for some others, we added a small text to explain the types of purposes a model can have to enhance the user experience in case they are not familiar with the literature behind ODD. Even though the model purpose guides the rest of the elements, it is not formal and produces no code.

We formalize our simulation strategies to further expose how we have approached the model purpose. We distinguish two categories

Design Concepts

Rationales

Rationales for Entities, Attributes, and Scales

Rationales for entities

Entity student **describes** university students
is used in forced-interaction, free-interaction, evaluate-interaction, learn-about, positive-interaction?
has rationale based on Adhoc rules
We chose to include only students because we are interested in the marginalization that emerges from the interactions among students and not as a result of student -teachers relationships for example.
Network university-relationship **describes** students have relationships with other students , the relationships are 2 directionals (are different in each direction)
is used in
has rationale based on Adhoc rules
Human relationships are two directional. One agent could have attraction A for another agent but the other agent may not return the same sentiment. University relations are relations build on the university interactions including acquaintances to study partners to friends.

Figure 5: Derived information - procedures for entities in the section Design Concepts.

Interaction forced-interaction describes interactions in classroom settings
The interaction involves a first_student **of type** entity student and a second_student **of type** entity student
Perform the interaction learn-about with first_student **and** second_student
Perform the interaction learn-about with second_student **and** first_student
Perform the interaction evaluate-interaction with first_student **and** second_student
Perform the interaction evaluate-interaction with second_student **and** first_student
<options for forced-interaction >

Figure 6: Procedure forced-interaction in the Section Process Overview and Scheduling.

Scheduling

1. Perform the interaction forced-interaction with select 50 % elements from entity student at anywhere
2. Perform the interaction free-interaction with select 50 % elements from entity student at anywhere

Press enter to add another step to the schedule

Figure 7: Section Scheduling in the tool.

of simulations based on the level of user engagement throughout the simulation: the Manual Experiments and the Automatic Experiments. In Manual Experiments, the user is active throughout the simulation. For each Manual Experiment, the user can adjust parameter values and start and stop the simulation based on their judgment. To do the manual experimentation, the user interacts with the user interface of the NetLogo platform. FODD automatically generates the features of the user interface with specifications from the **Manual Experiments** section.

Setting up Initialization values is to ensure that the active values in the simulation model are acknowledged and not defaulted by the platform. Aside from the values, the user specifies which parameters can be manipulated in the Manual Experiments, the appearance of the user interface in NetLogo, and the data visualization. FODD allows three methods for manipulating parameters during the Manual Experiments: Initialization with user input (a slider or a switch, for numerical and boolean values respectively), random Initialization using a random distribution, and Initialization with a fixed value given as an expression. The **Appearance** component of the section **Manual Experiments** sets the aesthetics and practicalities of the visual element in NetLogo, such as the grid size. The visualisation is done with tools such as graphs and counters. The Manual Experiment could relate to model exploration processes performed before the verification checks.

In Automatic Experiments, the user will design the experiments and the range of parameter values and then expect the platform or software to run the simulations without other input—the Automatic Experiments specified in the section **Experiments** of FODD.

Some platforms, including NetLogo, devote features to group multiple experiments and thus facilitate a user-friendly simulation design. The new section aims to reproduce this aspect. The section **Experiments** is comprised of subsections. The data collection subsection defines the values we want to collect for data analysis. The values can be any model attribute, aggregate statistic, or synthetic value. In the subsection **Experiments**, the user defines which parameters change values throughout the Experiment, the values they take, and the repetitions of each Experiment.

We argue that the experiment design should be explicit in the ODD formal document as it can show the extent to which and by what means the model purpose is explored. As discussed, a single purpose may require different experiments. Experiments differ in which parameters vary and in model appearance.

5 CURRENT STATUS AND NEXT STEPS

We have developed a proof-of-concept implementation of FODD that demonstrates the feasibility of our approach, available on GitHub [8], which enables experienced modelers to write and edit ODD-structured specifications for ABMs and automatically generate NetLogo code while maintaining aligned documentation.

We evaluate our progress along three dimensions: executability, expressivity, and user-friendliness. For executability, we developed a stable core architecture and progressively extended language structures, constraints, and transformations, enabling deterministic generation of working NetLogo code from checked specifications. In terms of expressivity, the current implementation captures six diverse models, including the MARG model (Figures 3–7), DomWorld [17, 18], and NetLogo library models (Voting, Fire, Wolf Sheep Predation, Cooperation). In principle, FODD can be extended to cover NetLogo's full computational expressiveness, but the current prototype intentionally restricts behavior constructs to preserve a higher abstraction level. Expanding the behavior language without exposing low-level NetLogo commands is planned future work. Our evaluation is preliminary and focuses on feasibility across six models, with completeness and expressivity trade-offs summarized concisely. For the evaluated models, the NetLogo implementation is generated entirely from the FODD specifications, meaning that no manual NetLogo coding is required beyond running the produced model. The DomWorld FODD description and generated NetLogo code are provided in [7] to illustrate the relationship between FODD and NetLogo code. For user-friendliness, ODD2NetLogo provides an editor that mirrors the ODD protocol structure and guides users with drop-down menus, templates, and error feedback (e.g., red underlines for invalid input), which can reduce cognitive load for code-level concerns by removing the need to reason in NetLogo or manually keep documentation and code aligned. However, initial evaluations reveal that the behavior specification language is not yet intuitive for social scientists. Overall, trade-offs include additional upfront specification effort compared to narrative ODD, a behavior language that still needs higher-level abstractions, a single NetLogo generation target today, and restrictions chosen to preserve abstraction and deterministic generation.

Building on this foundation, we are extending FODD along all three dimensions. To enhance executability, we continue to refine our agile development process, expanding the set of primitive activities and language structures to accommodate more diverse modeling patterns. To improve expressivity, we are broadening the range of model behaviors that can be formally specified while maintaining a conceptual level and avoiding low-level NetLogo commands, moving activity specifications from approximations toward more precise conceptual representations while completing the automation of derived sections in the Design Concepts element. To advance user-friendliness, we are redesigning the behavior specification interface to be more intuitive for social scientists and domain experts who may lack programming expertise. Our vision is to enable domain experts to first read and understand formal model specifications, and eventually author their own social simulation models. These extensions are being informed by ongoing feedback sessions with researchers from within and beyond the Social Simulation community.

By progressively extending FODD's executability, expressivity, and user-friendliness while maintaining formal rigor, we aim to make agent-based modeling more accessible without sacrificing the verification benefits that formal specifications provide. Open-source code availability alone does not guarantee transparency or accessibility if stakeholders cannot meaningfully interpret or modify it. ODD2NetLogo's usability features, together with automatic code generation and enforced consistency between conceptual descriptions and executable code, can lower barriers for non-technical collaborators to inspect, discuss, and validate models.

6 LIMITATIONS

In this paper, we present our proof-of-concept implementation of FODD, and it is essential to acknowledge its limitations. Our current generator targets NetLogo, which is a domain-specific language for ABMs rather than a general-purpose language such as Java, implying potential design constraints. In addition, since we define the mappings from FODD to NetLogo, aspects of the resulting computation and code style are determined by our transformation rules. Consequently, the generated code may not show the same variety as hand-written models and will not be optimized in all cases. The current generator prioritizes correctness and traceability over optimization, and standard optimization techniques can be applied to the generated NetLogo code when needed. Third, users must learn how to write FODD specifications to create model descriptions. Finally, FODD does not support reverse generation from code to specifications, so existing models must be described in FODD to enable extensions and updates.

7 SUMMARY

A FODD is a regular ODD protocol with verified links to the model code and requirements for complete descriptions of the different ODD features. We argue that using a FODD enhances model transparency and readability and provides benefits for participatory modeling approaches by giving more control to stakeholders. We have implemented the FODD using Domain-Specific Modeling and extended the existing ODD sections by including model exploration (Manual Experiments) and experimentation (Automatic Experiments) features. Additional benefits of our approach include the potential to make the model creation process more efficient, reduce documentation drift, clarify model concepts, and support evaluation of whether the model fulfills its intended purpose based on the experiments conducted. Our current implementation is in a proof-of-concept state and our tools are available online with open access.

ACKNOWLEDGMENTS

We are grateful to the students of the IKT445 Generative Programming course at UiA who provided feedback and support for the ODD2NetLogo tool, and to those who enhanced the tool through their project work.

REFERENCES

[1] Mohamed Abdou, Lynne Hamill, and Nigel Gilbert. 2012. *Designing and Building an Agent-Based Model.* Springer Netherlands, Dordrecht, 141–165. https://doi.org/10.1007/978-90-481-8927-4_8

[2] Petra Ahrweiler and Nigel Gilbert. 2015. *The Quality of Social Simulation: An Example from Research Policy Modelling.* Springer International Publishing, Cham, 35–55. https://doi.org/10.1007/978-3-319-12784-2_3

[3] Zenith Arnejo, Benoit Gaudou, Mehdi Saqalli, and Nathaniel Bantayan. 2025. Communicating Agent-Based Models to Stakeholders: A Scoping Review. *Journal of Artificial Societies and Social Simulation* 28, 2 (2025), 2. https://doi.org/10.18564/jasss.5623

[4] Olivier Barreteau, Pieter Bots, Katherine Daniell, Michel Etienne, Pascal Perez, Cécile Barnaud, Didier Bazile, Nicolas Becu, Jean-Christophe Castella, William's Daré, and Guy Trebuil. 2013. *Participatory Approaches.* Springer Berlin Heidelberg, Berlin, Heidelberg, 197–234. https://doi.org/10.1007/978-3-540-93813-2_10

[5] Michael Belfrage, Fabian Lorig, Christopher Frantz, Jason Tucker, and Paul Davidsson. 2025. The Transparency Imperative: The Need for Model Documentation for Engaging With Public Policy Following the EU AI Act. In *2025 Annual Modeling and Simulation Conference (ANNSIM).* 1–13.

[6] Tony Clark, Paul Sammut, and James Willans. 2015. Applied Metamodelling: A Foundation for Language Driven Development (Third Edition). https://doi.org/10.48550/ARXIV.1505.00149

[7] Multiple contributors. 2018-2026. DomWorld FODD and code (folder in uiano/odd2netlogo). GitHub folder. https://github.com/uiano/odd2netlogo/tree/master/Documentation/DomWorldFODDandcode

[8] Multiple contributors. 2018-2026. ODD2NetLogo: Generate NetLogo code from an ODD specification. GitHub repository. https://github.com/uiano/odd2netlogo

[9] Bruce Edmonds, Christophe Le Page, Mike Bithell, Edmund Chattoe-Brown, Volker Grimm, Ruth Meyer, Cristina Montañola Sales, Paul Ormerod, Hilton Root, and Flaminio Squazzoni. 2019. Different Modelling Purposes. *Journal of Artificial Societies and Social Simulation* 22, 3 (2019), 6. https://doi.org/10.18564/jasss.3993

[10] Bruce Edmonds and Lia ní Aodha. 2019. Using Agent-Based Modelling to Inform Policy – What Could Possibly Go Wrong?. In *Multi-Agent-Based Simulation XIX*, Paul Davidsson and Harko Verhagen (Eds.). Springer International Publishing, Cham, 1–16.

[11] Joshua M. Epstein. 2008. Why Model? *Journal of Artificial Societies and Social Simulation* 11, 4 (2008), 12. http://jasss.soc.surrey.ac.uk/11/4/12.html

[12] José Manuel Galán, Luis R. Izquierdo, Segismundo S. Izquierdo, José Ignacio Santos, Ricardo del Olmo, Adolfo López-Paredes, and Bruce Edmonds. 2009. Errors and Artefacts in Agent-Based Modelling. *Journal of Artificial Societies and Social Simulation* 12, 1 (2009), 1. https://www.jasss.org/12/1/1.html

[13] Claudius Gräbner. 2018. How to relate models to reality? An epistemological framework for the validation and verification of computational models. *Journal of Artificial Societies and Social Simulation* 21, 3 (2018).

[14] Volker Grimm, Uta Berger, Finn Bastiansen, Sigrunn Eliassen, Vincent Ginot, Jarl Giske, John Goss-Custard, Tamara Grand, Simone K. Heinz, Geir Huse, Andreas Huth, Jane U. Jepsen, Christian Jørgensen, Wolf M. Mooij, Birgit Müller, Guy Pe'er, Cyril Piou, Steven F. Railsback, Andrew M. Robbins, Martha M. Robbins, Eva Rossmanith, Nadja Rüger, Espen Strand, Sami Souissi, Richard A. Stillman, Rune Vabø, Ute Visser, and Donald L. DeAngelis. 2006. A standard protocol for describing individual-based and agent-based models. *Ecological Modelling* 198, 1 (2006), 115 – 126. https://doi.org/10.1016/j.ecolmodel.2006.04.023

[15] Volker Grimm, Steven F. Railsback, Christian E. Vincenot, Uta Berger, Cara Gallagher, Donald L. DeAngelis, Bruce Edmonds, Jiaqi Ge, Jarl Giske, Jürgen Groeneveld, Alice S.A. Johnston, Alexander Milles, Jacob Nabe-Nielsen, J. Gareth Polhill, Viktoriia Radchuk, Marie-Sophie Rohwäder, Richard A. Stillman, Jan C. Thiele, and Daniel Ayllón. 2020. The ODD Protocol for Describing Agent-Based and Other Simulation Models: A Second Update to Improve Clarity, Replication, and Structural Realism. *Journal of Artificial Societies and Social Simulation* 23, 2 (2020), 7. https://doi.org/10.18564/jasss.4259

[16] Amy Heather, Thomas Monks, Alison Harper, Navonil Mustafee, and Andrew Mayne. 2025. On the reproducibility of discrete-event simulation studies in health research: an empirical study using open models. *Journal of Simulation* (2025), 1–25. https://doi.org/10.1080/17477778.2025.2552177

[17] Charlotte Hemelrijk. 2002. Despotic societies, sexual attraction and the emergence of male 'tolerance': an agent-based model. *Behaviour* 139, 6 (2002), 729–747.

[18] Charlotte K. Hemelrijk. 1998. Risk Sensitive and Ambiguity Reducing Dominance Interactions in a Virtual Laboratory. In *From Animals to Animats 5: Proceedings of the Fifth International Conference on Simulation of Adaptive Behavior.* The MIT Press. https://doi.org/10.7551/mitpress/3119.003.0040 arXiv:https://direct.mit.edu/book/chapter-pdf/2311019/9780262291385_cbm.pdf

[19] JetBrains. 2021. MPS Meta Programming System Version 2021.2. https://www.jetbrains.com/mps/

[20] Steven Kelly and Juha-Pekka Tolvanen. 2008. *Domain-specific modeling : enabling full code generation.* Wiley, Hoboken, N.J.

[21] Tatiane Micheletti, Marie-Christin Wimmler, Uta Berger, Volker Grimm, and Eliot J. McIntire. 2024. Beyond guides, protocols and acronyms: Adoption of good modelling practices depends on challenging academia's status quo in ecology. *Ecological Modelling* 496 (2024), 110829. https://doi.org/10.1016/j.ecolmodel.2024.110829

[22] Daniel Moyo, Abdallah K. Ally, Alan Brennan, Paul Norman, Robin C. Purshouse, and Mark Strong. 2015. Agile Development of an Attitude-Behaviour Driven Simulation of Alcohol Consumption Dynamics. *Journal of Artificial Societies and Social Simulation* 18, 3 (2015), 10. https://doi.org/10.18564/jasss.2841

[23] Birgit Müller, Friedrich Bohn, Gunnar Dreßler, Jürgen Groeneveld, Christian Klassert, Romina Martin, Maja Schlüter, Jule Schulze, Hanna Weise, and Nina Schwarz. 2013. Describing human decisions in agent-based models – ODD + D, an extension of the ODD protocol. *Environmental Modelling & Software* 48 (2013), 37 – 48. https://doi.org/10.1016/j.envsoft.2013.06.003

[24] J. Gary Polhill and Nicholas M. Gotts. 2009. Ontologies for transparent integrated human-natural system modelling. *Landscape Ecology* 24, 9 (2009), 1255. https://doi.org/10.1007/s10980-009-9381-5

[25] Andrew D. Selbst, Danah Boyd, Sorelle A. Friedler, Suresh Venkatasubramanian, and Janet Vertesi. 2019. Fairness and Abstraction in Sociotechnical Systems. In *Proceedings of the Conference on Fairness, Accountability, and Transparency* (Atlanta, GA, USA) *(FAT* '19).* Association for Computing Machinery, New York, NY, USA, 59–68. https://doi.org/10.1145/3287560.3287598

[26] Flaminio Squazzoni, J. Gareth Polhill, Bruce Edmonds, Petra Ahrweiler, Patrycja Antosz, Geeske Scholz, Emile Chappin, Melania Borit, Harko Verhagen, Francesca Giardini, and Nigel Gilbert. 2020. Computational Models That Matter During a Global Pandemic Outbreak: A Call to Action. *Journal of Artificial Societies and Social Simulation* 23, 2 (2020), 10. https://doi.org/10.18564/jasss.4298

[27] Themis Dimitra Xanthopoulou. 2020. MARG: NetLogo model and ODD documentation. GitHub repository. https://github.com/themisdx/MARG

[28] Connie Wang, Bin-Tzong Chi, and Shu-Heng Chen. 2019. Agent-based model for centralized student admission process. https://www.comses.net/codebases/4774/releases/1.0.0/

[29] Uri Wilensky. 1999. *NetLogo Home Page.* Web Page. Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL. http://ccl.northwestern.edu/netlogo/

[30] Themis Dimitra Xanthopoulou, Andreas Prinz, and F. LeRon Shults. 2019. Generating Executable Code from High-Level Social or Socio-Ecological Model Descriptions. In *System Analysis and Modeling. Languages, Methods, and Tools for Industry 4.0*, Pau Fonseca i Casas, Maria-Ribera Sancho, and Edel Sherratt (Eds.). Springer International Publishing, Cham, 150–162.

[31] Themis Dimitra Xanthopoulou, Andreas Prinz, and F. LeRon Shults. 2022. The Problem with Bullying: Lessons Learned from Modelling Marginalization with Diverse Stakeholders. In *Advances in Social Simulation*, Marcin Czupryna and Bogumił Kamiński (Eds.). Springer International Publishing, 289–300.

[32] Themis Dimitra Xanthopoulou, Ivan Puga-Gonzalez, F. LeRon Shults, and Andreas Prinz. 2020. Modeling marginalization: emergence, social physics, and social ethics of bullying. In *Proceedings of the 2020 Spring Simulation Conference* (Fairfax, Virginia) *(SpringSim '20).* Society for Computer Simulation International, San Diego, CA, USA, Article 40, 12 pages.