A Methodology for DSML-assisted Participatory Agent-Based Enterprise Modelling *

Thomas Godfrey[1][0000−0002−9904−0744], Rahul Batra[2], Sam Douthwaite[2][0000−0003−4469−7623], Jonathan Edgeworth[2], Simon Miles[1][0000−0001−9956−4217], and Steffen Zschaler[1][0000−0001−9062−6637]

[1] Department of Informatics, King's College London, Bush House, Strand Campus, 30, Aldwych, London, WC2B 4BG, UK
{thomas.1.godfrey,simon.miles,steffen.zschaler}@kcl.ac.uk
[2] Guy's and St Thomas' NHS Foundation Trust, Westminster Bridge Rd, London SE1 7EH, UK
{Rahul.Batra1,Sam.Douthwaite,jonathan.edgeworth}@gstt.nhs.uk

Abstract. Participatory agent-based modelling (ABM) can help bring the benefits of simulation to domain users by actively involving stakeholders in the development process. Collaboration in enterprise modelling can improve the model developer's understanding of the domain and therefore improve the effectiveness of domain analysis. Where many agent-oriented methodologies focus on the development of one-off models, domain-specific modelling languages (DSML) can improve the re-use of concepts identified in domain analysis across multiple case studies and expose modelling concepts in domain-appropriate terms, increasing model accessibility. To realise the benefits of DSMLs we need to understand how DSML development can be incorporated into typical agent-based modelling. In this paper we discuss existing methodologies for ABM development and DSML development, and we discuss the benefits merging the two can bring. We present a methodology for DSML-assisted participatory agent-based modelling, and support the methodology with a case study— a modelling exercise conducted in collaboration with a hospital emergency department on the topic of infection control for COVID-19 and Influenza.

Keywords: Agent-Based Modelling · Participatory Modelling · Domain-Specific Modelling.

1 Introduction

Agent-based modelling (ABM) can be an attractive option for enterprise modelling due to the capacity of agent-based models to capture: high levels of heterogeneity, organic interactions between model entities, and organic emergent behaviours [7]. To develop effective agent-based models, it is important that

* Thomas Godfrey is supported by the KCL funded Centre for Doctoral Training (CDT) in Data-Driven Health.

model developers have a strong understanding of the domain. Developers need to identify the actors involved in a process, their behaviours, and how they interact between themselves and the environment. These behaviours need to be translated into a set of rules to model but determining *which* actors and actions should be captured and *how* they should be modelled can be a complex process. Participatory modelling [28] encourages close collaboration between domain stakeholders and model developers, improving the quality of domain analysis. However, existing approaches to participatory modelling typically depend on general-purpose, or agent-oriented, languages that require the model developer to translate design concepts into a technical implementation. This abstraction process is manual, is prone to error and ambiguities, and presents a lack of re-use.

Domain-specific modelling languages (DSMLs) [9] are an attractive tool for supporting participatory modelling. DSMLs consist of a concrete syntax (the language grammar), an abstract syntax (a meta-model of the language structure), and semantics (the 'meaning' of the language concepts). DSMLs expose domain-appropriate concepts in the concrete syntax, allowing the user to utilise these high-level terms to implement models. This can allow domain stakeholders to collaborate on the model implementation directly, reducing the risk of abstraction errors. Models written in a DSML are typically automatically generated into computer-readable code, allowing for traceability of the model, and efficiency of development. In developing a DSML, the developer performs domain engineering [23] by identifying the common and variable properties of the domain and formalising them in the language syntax. The DSML can then be used to generate specific model instances by expressing the particulars of the model using the language concepts (known as application engineering). The meta-model of the DSML effectively provides an (evolving) domain model for future modelling exercises, improving the re-use of concepts for specific applications.

Existing approaches to DSML-assisted modelling methodologies place the abstraction level of the DSML at the ABM platform level (such as easyABMS [11], and the work of Stark and Barn [2]) or use technically complex languages [14]. Other approaches exist that use high domain-level concepts such as MAIA [12] and INGENIAS [22] for developing social science simulations. However, these works provide a generic language that developers need to learn how to instantiate correctly. In this paper we present a methodology for creating DSMLs for ABM development in a domain-agnostic way such that developers can adapt a language for their particular domain. We refer to these languages as 'high-level DSMLs'. We take inspiration from the vision of DSMLs for ABM development outlined by Zschaler and Polack [30], and the principles of effective participatory modelling detailed in the CoSMoS project [27]. We make the following contributions:

1. A novel methodology for DSML-assisted participatory agent-based modelling
2. A review of current approaches to participatory ABM and DSML development
3. An ABM case study on infection control in a hospital emergency department to motivate and illustrate our methodology

In section 2, we introduce a participatory modelling case study conducted in collaboration with St Thomas' Hospital Emergency Department in London. We use this case study to motivate the need for a methodology for DSML-assisted participatory modelling. We discuss some existing methodologies for participatory agent-based modelling and domain-specific modelling languages in sections 3 and 4. In section 5, we use the case study to motivate our methodology for DSML-assisted agent-based modelling.

2 Case study

In this section we detail the case study for our methodology —a modelling case study conducted in the Spring of 2022 in collaboration with St. Thomas' Hospital emergency department (ED) and infection control teams. The intention of the study was to analyse the infection control measures for COVID-19 and Influenza introduced in the hospital emergency department during the Winter of 2021. An ABM was developed for this study, using a healthcare-focused DSML. Below we provide background information to the case study, and the motivation for developing a participatory modelling methodology.

At St. Thomas' Hospital, the infection control team are in regular contact with the ED to help anticipate, and mitigate, infection control problems. The product of these meetings is typically a policy-revision for hospital staff. New policies define how staff should manage patient care such as what tests they should perform on patients, how to interpret test results, and where patients should be admitted to according to their infection status [19]. In St. Thomas' Hospital, these policies are communicated to staff through 'action cards' which are flowchart-like process descriptions that are available on the trust intranet. As new policies are introduced, these action cards will be updated and replaced. An example used during a peak of COVID-19 prevalence in 2020 can be found in figure 1.

During the Winter of 2021, infection control were cautious about the potential return of Influenza as well as the already expected COVID-19. This would introduce further complications for infection control as staff would need to mitigate the risk of cross-infection between patients with different infectious diseases. This required new rules on patient isolation, and triggered the introduction of new rapid tests for Influenza and COVID-19. As part of this planning, a series of new action cards were developed using our DSML and discussed between the infection control and ED teams. In the spring of 2022, we modelled the processes defined in these action cards and evaluated directional changes in infection control metrics between the different action cards including: infection risk (measured by the number of infected patients admitted to a non-infectious cohort) and resource usage (measured as number of tests used within a particular time-frame).

During the study, we identified some key areas where typical agent-oriented methodologies are not suitable for the healthcare domain. In a hospital environment, staff need to make decisions quickly to keep up with the rapid needs of patient care. Restrictions on time can often make typical ABM develop-

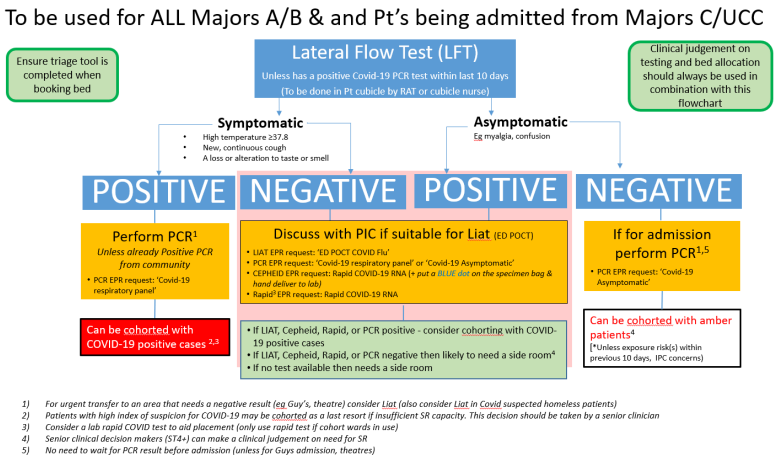

Fig. 1. Action card example.

ment infeasible due to their time and resource requirements. Strong domain analysis often requires the developer to spend considerable time within, and learning about, the domain. Typical agent-oriented methodologies do not formalise the findings of domain analysis into re-usable concepts, instead focusing on application-engineering for a particular case study. This results in poor re-use of domain concepts in the modelling process and additional development overheads for subsequent modelling exercises. Every time a domain process changes and every time a new model is developed, the developers need to restart the model conceptualisation and design steps. With a DSML, we expected domain analysis to contribute additively to the language meta-model. As more models are developed, we expected the scope and specificity of the language to improve for the target domain. When specific models are developed in the future, there should be less requirement to complete any further domain analysis, and instead the DSML will already provide the necessary concepts to instantiate. This subsequently makes the model development process less cumbersome on stakeholders and developers over time.

Hospital EDs are continually evolving to keep up with demands. In our example above, the series of action card iterations introduced both variations on existing domain processes, as well as the introduction of entirely new processes. In a typical agent-based modelling approach, making changes such as these to the model can be complex and unwieldy. The definition of high-level domain processes is distributed across individual agent and environment definitions, making it challenging to identify and refactor the model implementation. The model developer (in collaboration with domain stakeholders) would need to decompose each action card into distinct agent definitions, where each agent has their own list of behaviours representing part of the overall action card process. If an alternative action card is to be tested, the model developer would need to repeat

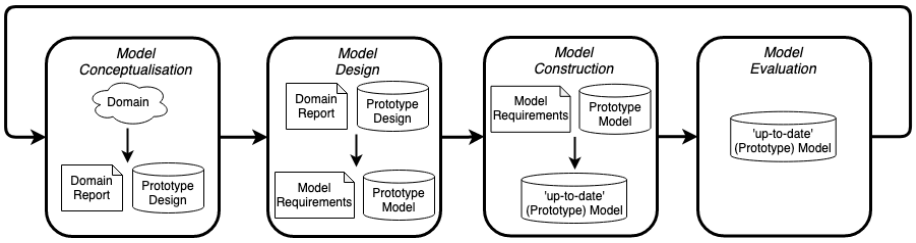

Fig. 2. The ABM development cycle.

this decomposition process again or attempt to refactor the existing agent implementations. This activity is manual, time-consuming, and error-prone. Using a DSML, we expected to largely avoid these issues by capturing high-level domain processes explicitly in the modelling language. With a high-level DSML, action cards can be modelled in one central viewpoint (with orthogonal high-level concerns captured in alternative modelling viewpoints). The decomposition of the global domain process into individual agents and their behaviours is completed automatically during code generation, making changes to domain processes easier and quicker to facilitate.

We have identified the potential for using DSMLs in model development, and how they can benefit collaboration with domain stakeholders and improve re-use in domain analysis. In the next sections, we discuss methodologies for how to develop DSMLs, and how DSML development can be incorporated into typical ABM methodologies. We start with a description of typical ABM methodologies.

3 Agent-based modelling methodologies

Ramanath and Gilbert [24] present a literature review on the topic of participatory ABM development and identify patterns in previous work on social simulations to develop a methodology for participatory modelling. They describe the participatory design process as consisting of 4 key phases which we present in figure 2 and discuss below:

The first phase is **model conceptualisation** in which the model developers consult domain stakeholders to learn about domain processes, the relevant domain problems, and the stakeholders' expectations for model outputs. This phase is conducted through the use of different example scenarios developed informally by the model developers. The model developers may conduct their own independent research in the relevant literature prior to this stage in order to form foundational knowledge of the domain.

The next phase is **model design**. This step involves taking the results of scenario analysis into a workshop setting with groups of domain stakeholders. The intention is to conceptualise and design the model with the end-users, sharing informal model prototypes and discussing model structure requirements. The aim of this process is to identify the main entities involved in domain processes

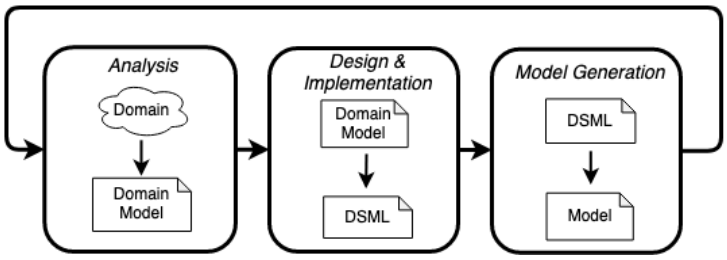

Fig. 3. The DSML development cycle.

(whether active or passive) and to identify their key attributes, behaviours, and interactions. Ideally, a diverse group of stakeholders should be present in these activities in order to contribute their respective background knowledge on domain processes to be implemented in the model.

The output of the model design phase is formalised during **model construction**. During this phase, a basic agent-based simulation is developed by the model developers with input from the domain stakeholders. The developers will translate the entities identified during model design into agents with their respective properties and behaviours. The simulation is not intended to be complete, but instead should provide a concrete software artifact for analysis in the next phase.

The **evaluation** phase is conducted with the stakeholders in the form of a user panel. This involves the model developers demonstrating the simulation to the stakeholders and discussing any ambiguities that remained during its implementation. For example, the model developers should clarify any issues related to agent behavioural rules, data availability, etc. that were not resolved during model construction. The user panel will provide an opportunity for the domain stakeholders to experience first-hand a version of the model and to provide feedback. Lee et al. [17], and Klügl [16] provide excellent resources on ABM output analysis and validation techniques. The feedback produced from these user panels can be used to inform changes to the model design for further iterations of development, starting again from **model conceptualisation**. Once the model is deemed fit for purpose, it can be used to conduct experiments to provide meaningful outputs to stakeholders.

4 DSML Methodologies

In this section, we discuss existing approaches to DSML development.

Mernik et al. [18] review a range of DSML-related papers and identify patterns in the development processes. They distil these patterns into a methodology covering all aspects of development from 'decision' of when to develop a DSML to 'deployment' of the language. For the scope of this paper we will focus on the intermediary steps of domain analysis, design and implementation, which we present in figure 3 and discuss below:

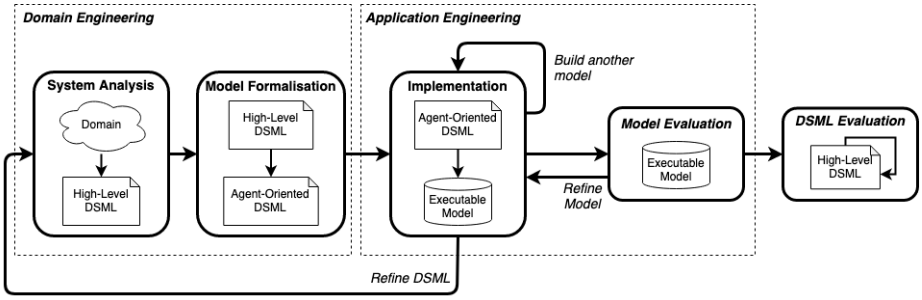

Fig. 4. The methodology cycle.

Analysis involves gathering knowledge about the domain and the scope of the problem(s) the users wish to address. The intention of this phase is to establish the terminology used in the domain problem, and the semantics of different domain concepts. These concepts will then be formalised in the next phase.

Next, the **design and implementation** phase then involves converting the domain model generated in the analysis phase into a DSML definition. This will include identifying which domain concepts are constant and and which are variable. We refer to Voelter [29], or Fowler [9] for fundamental concepts of DSML design.

Once the DSML has been developed, it can then be used in application engineering to generate specific model instances. We refer to this phase as **model generation**, which involves identifying which DSML concepts are needed for a particular model case study, and creating instantiations of those concepts.

5 A Methodology for DSML-assisted Participatory Agent-Based Modelling

In this section, we propose a methodology for DSML-assisted participatory agent-based modelling influenced by Mernik et al. [18] for the DSML-side, and Ramanath and Gilbert [24] for the ABM-side. We adopt the terminology of the CoSMoS approach [27] to describe the abstraction level of different models developed in the process. The *domain model* refers to a description of a domain problem as understood from observations etc. The *platform model* refers to the formal specification of the domain model including technical requirements for it's implementation. Finally, the *simulation model* is the concrete model implementation itself.

The methodology follows an iterative cycle of steps from system analysis to evaluation. A diagram of the methodology is shown in figure 4. The steps are divided into those that constitute domain engineering and those that constitute

application engineering via the dotted lines. The large boxes named system analysis, model formalisation, implementation, and model evaluation represent the ABM development steps. Inside the large boxes, we represent the DSML-related processes involved. For example, in system analysis, the domain is translated into a high-level DSML specification. In model formalisation, generation rules are added to the high-level DSML such that it can be translated into an agent-oriented DSML. Implementation then constitutes the application engineering process in which the DSML is instantiated for specific modelling case studies. At this stage, either the implementation step is repeated to represent development of multiple models, or if the DSML is not sufficient to specify a particular model, then there is a feedback loop to system analysis to show repetition of the domain engineering steps. The model(s) generated during implementation will then be validated [16]. At this stage, if the model is not fit for purpose, then there is a loop back to implementation such that the model (i.e. the DSML instantiation) can be refined. A final (optional) step is to evaluate the high-level DSML using comprehension and usability exercises. We discuss each step below:

5.1 System Analysis

During system analysis, the developer will seek to understand the domain to be modelled. This involves establishing what domain problems are to be addressed and what the criticality of those problems are. The criticality will depend on what the domain stakeholders wish to learn from the model [13]: Will the model be used to encourage *learning* between stakeholders? Will it be used to *explain* domain behaviours? Or will it be used to make *predictions* about future behaviours? Once the domain problem has been established, the problem should be decomposed into it's related domain processes and the stakeholders involved in those processes. These stakeholders should be consulted so that they can contribute their domain knowledge about the process from their respective backgrounds, viewpoints and vocabularies.

In a typical ABM methodology, this step involves the production of informal design artefacts to form a domain model. This domain model is constructed on a model-by-model basis with an application-engineering approach —focusing on identifying and recording domain processes for a particular domain problem. Using DSMLs introduces a shift towards a domain engineering approach. The DSML is not intended to be a single-use product, and instead should be used to construct a variety of models in the same domain. The DSML meta-model will need to be constructed as an ontology of re-usable, generic, domain concepts that can be instantiated for specific models as and when they are developed. The repeated exercise of domain engineering conducted during this step allows the DSML to mature and reduces the need for further domain analysis in future case studies.

For a brand-new DSML, domain analysis will involve *construction* of the language meta-model. The language developer will take the findings of the domain analysis step described above to identify the scope and nature of domain

concepts. These concepts will provide the basis for the language abstract syntax, and the vocabulary used by the stakeholders to describe those concepts will provide the basis for the language concrete syntax. For a more mature DSML, domain analysis should involve *maintenance* of the meta-model. The developer should supplement the DSML with new concepts, or refactor existing concepts, where appropriate.

Case Study Experience. In our healthcare case study, we identified that action cards (such as the one shown in figure 1) are a familiar notation for describing healthcare processes in our target hospital domain. While healthcare professionals may not think of action card development as 'modelling', action cards can be seen as informal 'grass-roots' models [25]. We took these pre-existing models as the basis for our own DSML design. An example of an action card written in our DSML is shown in figure 5.

For simpler processes, the action card DSML offered sufficient scope to model the domain processes our domain stakeholders were interested in. For example, to model the rapid COVID-19 test described in section 2, we simply used the DSML to define an *action* in an *action card* that made reference to the rapid *test*, which included a property for time *duration*. The tests are conducted at the bedside and results are available within a very short time period. However, we later needed to make additions to the language for more complex processes —for example, the introduction of a point-of-care test called a LIAT [4]. The LIAT requires staff to swab the patient, and then transport the sample to an analyser machine which can process the sample and return a result within 30 minutes. Because of the longer execution time, and the more complex testing process, our DSML did not expose a suitable level of granularity to accurately model this type of test.

In identifying this gap, we needed to update our DSML design. We added a new language concept to represent a *testing process* — effectively a sub-process that can be defined for each type of test in the DSML which can be referenced by the existing action card concept. The DSML generator automatically weaves the testing process definition into the action card where appropriate. While these processes are inter-dependent, they can be captured in their own distinct viewpoint and vocabulary in the DSML allowing for a multi-view modelling approach to model development [5].

5.2 Model Formalisation

From the previous step, a domain problem has been identified and represented using a high-level DSML. In typical participatory ABM methodologies, the next step would be to translate the domain model into a platform model (i.e. translate the informal design artefacts into an agent-oriented formalisation). However, rather than creating a domain model for a singular problem, we have developed a DSML suitable for capturing a variety of domain problems. The next stage, therefore, is to establish how these DSML concepts can be translated into an

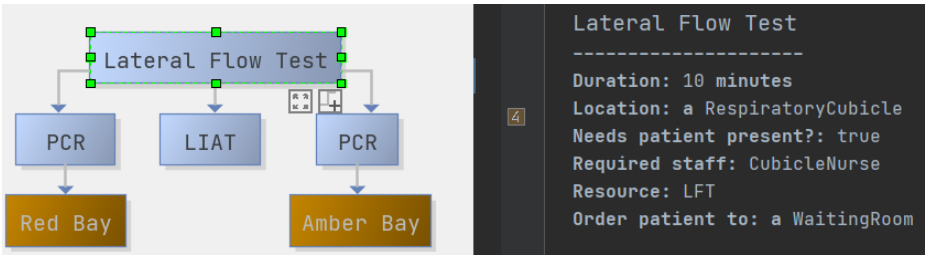

Fig. 5. Action card concrete syntax.

executable model. Primarily, this will involve implementing generation rules for each domain concept added to the DSML during the system analysis step. The developer will need to consider how these new generation rules will interact with each other, and with the concepts that already exist in the language from previous development cycles, if any.

In line with [30], we do not translate domain-level concepts directly into a platform model written in a general-purpose language, but instead develop an internal agent-oriented DSML to sub-divide the code generation process. Either existing agent-oriented language can be adopted (such as DSL4ABMS [26], ReLogo [21], or ESL [6]) or the language developer can implement their own. The developer, in collaboration with domain stakeholders, will then determine how the high-level DSML concepts should translate into ABM concepts in the agent-oriented language. In collaborating on developing these generation rules, the developers explicitly encode knowledge about the domain that may otherwise remain implicit. As the DSMLs mature and less language refactoring is required, this step should become simpler and quicker to complete.

The benefits of dividing the DSML generation in this way include: improving model traceability, reducing the risk of abstraction errors, and allowing for modularity in the implementation technique of the model. For example, while in the current work we focus on agent-based modelling, it is possible (and sometimes desirable [8]) to use alternative modelling techniques such as system dynamics, Monte Carlo modelling etc. [15] either alongside, or in place of, agent-based modelling. Alternative internal DSMLs can be developed and 'swapped-in' to provide the specific implementation details.

Case Study Experience. From our own experience, we identified action cards as an important concept during system analysis and so designed our DSML such that action cards could be encoded explicitly in the language, using graphical language elements. This high-level DSML is then automatically generated into an agent-oriented internal DSML as discussed above. This allowed for a separation of concerns, simplifying the generators for the DSML by splitting the generation process in two. The structure of the DSML is shown in figure 6.

Figure 7 shows a sample of the generated actor language code. Specifically shown is the actor rules for the 'lateral flow test' action, detailing how an agent

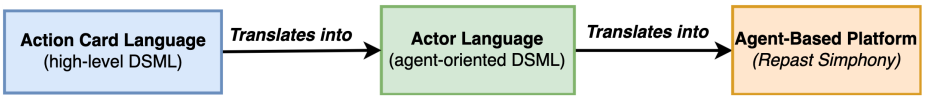

Fig. 6. The DSML Structure.

```
Behaviour: LateralFlowTest
  Trigger Message: NewpatientArrive
  Steps:
  --------------------------
    go to the patient <no description>
    ---------------------
    ask the patient to follow me <no description>
    ---------------------
    go to a RespiratoryCubicle <no description>
    ---------------------
    use LFD in the patient
    ---------------------
    ask the patient to go to a WaitingRoom <no description>
```

Fig. 7. Actor language concrete syntax.

should administer a lateral flow COVID-19 test (LFD) on a patient. Shown is the trigger for this behaviour (in this case, the patient arriving in the department), and the sub-steps for the action including taking the patient to a respiratory cubicle, using the LFD on the patient, and asking the patient to go back to the waiting room.

5.3 Implementation

During implementation, the DSML will then be used to specify a specific model instance. The language concepts will be used to construct the model which will get automatically generated into computer-readable code according to the generation rules implemented in the previous step. If the DSML is not sufficient to express a particular model, then the development cycle should be restarted from the 'system analysis' step (see feedback loop 'refine DSML' in figure 4).

Case Study Experience. We include another feedback loop for 'build another model'. For example, as discussed in our case study, we wished to model not just a single action card but to compare a series of different action cards. For example, during our case study one alternative action card we wished to model involved adding a test for whether the patient has had a recent contact with a COVID-19 infectious person, and a workflow for deciding whether the patient should be admitted to a side room or not. By this stage of development, we were able to implement alternative action cards like these without making further changes to the DSML.

Once the developer has produced the models they are interested in, they can be automatically generated into code and then evaluated in the next step.

5.4 Evaluation

Model Evaluation. The generated model can now be calibrated and evaluated. Calibration involves testing the model under different parameter values and updating these parameter values until model outputs are representative of real-world data [17]. Evaluation then involves using the calibrated parameter values to check that the model behaviour matches real-world observations and data under different concrete scenarios [16]. We note that the level of detail required during evaluation should depend on the criticality of the domain problem as established during system analysis. If the model is to be used for prediction, then it is especially important that the model is evaluated to a high standard. There is less requirement for detailed evaluation if the model is to be used for explanation, and less so again for learning. There should be an appropriate balance between a well-evaluated model and a model that can produce results more quickly. If the model is deemed unfit for purpose then the developer should return to the implementation step to check that the model has been implemented correctly in the DSML.

DSML Evaluation (optional). Primarily, the DSML will be evaluated naturally via completion of the development cycle, specifically during the 'refine DSML' feedback loop and via repeated modelling exercises. The system analysis step will highlight any issues in language coverage via missing concepts in the DSML syntax which can be added where appropriate. However, if time permits, the DSML can be evaluated more formally according to the language comprehensibility and usability. To evaluate the comprehensibility of the DSML, we refer to Moody's 'cognitive framework' [20] metrics, and for usability evaluation, we refer to Barišić et al. [3] and Alaca et al. [1].

Case Study Experience. As part of our healthcare case study we found that the DSML aided parts of the evaluation steps. Because the model design artefacts were explicitly and formally encoded in the language, we were able to more easily conduct face validity with the domain stakeholders. We could refer directly to the model definition in the DSML during model execution, rather than using more informal design documents. For example, we were able to discuss the definition of an action card in our DSML with staff from the ED and infection control teams at St. Thomas' Hospital. Even before executing the model, we were able to discuss and resolve any ambiguities in the modelling requirements. The staff were able to identify errors in the action card definition, but were also able to identify potential practical issues with the action card in vivo and were able to suggest real-world changes to the action card that could be investigated. Our interaction indicated that the healthcare staff were able to understand the action card DSML without significant previous experience, and they were able to use

the model to identify and discuss issues with real-world clinical practise even before execution.

Upon executing the model, the stakeholders could then observe the different staff and patient agents moving through a representation of the hospital wards in a graphical interface. Simultaneously, we displayed the action card definition implemented using our high-level DSML. This allowed us to directly relate properties of the model execution to the related concepts in the DSML, encouraging discussion with the stakeholders about potential errors. Of any errors, either the model definition was wrong and so the model should be updated in the DSML, or the model had not been implemented correctly and so the DSML itself should be inspected (see feedback loops for 'refine model' and 'refine DSML' in figure 4).

It should be noted, however, that using DSMLs did not significantly impact model calibration. This is a process that relies on statistical comparison of model outputs with real-world data and is typically conducted through the use of statistical software packages (such as RRepast [10]). These tools usually require the developer to define scripts for data analysis, and for model inputs and outputs to be in particular formats. The only benefit of our DSML was that it could generate the relevant scripts and fulfil the relevant formatting requirements during model generation.

6 Conclusions

We motivated our work by a participatory modelling exercise conducted in the domain of infection control in emergency care. We intend for our methodology to reflect the practical demands of participatory modelling, including the limited capacity for stakeholder involvement and the need for rapid model results. While we initially assumed that the benefit of DSMLs would be the ability for domain stakeholders to directly interact with model implementation, we found that the primary benefits were the promotion of communication between stakeholders on domain problems, and the re-use of domain concepts for improving the pace of system analysis. Our future work will focus on evaluating the methodology through further modelling case studies. Open research topics include investigating the use of bi-directional feedback of model execution to DSML design as discussed by Zschaler and Polack [30] and the full integration of model and domain through a digital twin as discussed by Barat et al. [2].

References

1. Alaca, O.F., Tezel, B.T., Challenger, M., Goulão, M., Amaral, V., Kardas, G.: AgentDSM-Eval: A framework for the evaluation of domain-specific modeling languages for multi-agent systems. Computer Standards & Interfaces **76**, 103513 (2021). https://doi.org/https://doi.org/10.1016/j.csi.2021.103513, https://www.sciencedirect.com/science/article/pii/S0920548921000088
2. Barat, S., Kulkarni, V., Clark, T., Barn, B.: An actor based simulation driven digital twin for analyzing complex business systems.

In: 2019 Winter Simulation Conference (WSC). pp. 157–168 (2019). https://doi.org/10.1109/WSC40007.2019.9004694
3. Barišić, A., Amaral, V., Goulao, M., Barroca, B.: Quality in use of domain-specific languages: a case study. In: Proceedings of the 3rd ACM SIGPLAN Workshop on Evaluation and Usability of Programming Languages and Tools. pp. 65–72 (2011)
4. Blackall, D., Moreno, R., Jin, J., Plotinsky, R., Dworkin, R., Oethinger, M.: Performance Characteristics of the Roche Diagnostics cobas Liat PCR System as a COVID-19 Screening Tool for Hospital Admissions in a Regional Health Care Delivery System. Journal of clinical microbiology **59**(10), e01278–21 (2021)
5. Bork, D., Sinz, E.J.: Bridging the gap from a multi-view modelling method to the design of a multi-view modelling tool. Enterprise Modelling and Information Systems Architectures (EMISAJ) **8**(2), 25–41 (2013)
6. Clark, T., Kulkarni, V., Barat, S., Barn, B.: Esl: An actor-based platform for developing emergent behaviour organisation simulations. In: Demazeau, Y., Davidsson, P., Bajo, J., Vale, Z. (eds.) Advances in Practical Applications of Cyber-Physical Multi-Agent Systems: The PAAMS Collection. pp. 311–315. Springer International Publishing, Cham (2017)
7. Crooks, A.T., Heppenstall, A.J.: Introduction to agent-based modelling. In: Agent-based models of geographical systems, pp. 85–105. Springer (2012)
8. Fakhimi, M., Anagnostou, A., Stergioulas, L., Taylor, S.J.E.: A hybrid agent-based and discrete event simulation approach for sustainable strategic planning and simulation analytics. In: Proceedings of the Winter Simulation Conference 2014. pp. 1573–1584 (2014). https://doi.org/10.1109/WSC.2014.7020009
9. Fowler, M.: Domain Specific Languages. Addison-Wesley Professional, 1st edn. (2010)
10. García, A.P., Rodríguez-Patón, A.: Analyzing repast symphony models in r with rrepast package. bioRxiv p. 047985 (2016)
11. 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)
12. 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)
13. Heldal, R., Pelliccione, P., Eliasson, U., Lantz, J., Derehag, J., Whittle, J.: Descriptive vs prescriptive models in industry. In: Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems. p. 216–226. MODELS '16, Association for Computing Machinery, New York, NY, USA (2016). https://doi.org/10.1145/2976767.2976808, https://doi.org/10.1145/2976767.2976808
14. Iba, T., Matsuzawa, Y., Aoyama, N.: From conceptual models to simulation models: Model driven development of agent-based simulations. In: 9th Workshop on economics and heterogeneous interacting agents. vol. 28, p. 149. Citeseer (2004)
15. Katsaliaki, K., Mustafee, N.: Applications of simulation within the healthcare context. Journal of the operational research society **62**(8), 1431–1451 (2011)
16. Klügl, F.: A validation methodology for agent-based simulations. In: Proceedings of the 2008 ACM symposium on Applied computing. pp. 39–43 (2008)
17. Lee, J.S., Filatova, T., Ligmann-Zielinska, A., Hassani-Mahmooei, B., Stonedahl, F., Lorscheid, I., Voinov, A., Polhill, J.G., Sun, Z., Parker, D.C.: The complexities of agent-based modeling output analysis. Journal of Artificial Societies and Social Simulation **18**(4) (2015)

18. Mernik, M., Heering, J., Sloane, A.M.: When and how to develop domain-specific languages. ACM computing surveys (CSUR) **37**(4), 316–344 (2005)
19. Merrick, B., Noronha, M., Batra, R., Douthwaite, S., Nebbia, G., Snell, L., Pickering, S., Galao, R., Whitfield, J., Jahangeer, A., Gunawardena, R., Godfrey, T., Laifa, R., Webber, K., Cliff, P., Cunningham, E., Neil, S., Gettings, H., Edgeworth, J., Harrison, H.: Real-world deployment of lateral flow SARS-CoV-2 antigen detection in the emergency department to provide rapid, accurate and safe diagnosis of COVID-19. Infection Prevention in Practice **3**(4), 100186 (dec 2021). https://doi.org/10.1016/J.INFPIP.2021.100186, https://linkinghub.elsevier.com/retrieve/pii/S2590088921000755
20. Moody, D.: The "physics" of notations: toward a scientific basis for constructing visual notations in software engineering. IEEE Transactions on software engineering **35**(6), 756–779 (2009)
21. Ozik, J., Collier, N.T., Murphy, J.T., North, M.J.: The ReLogo agent-based modeling language. In: 2013 Winter Simulations Conference (WSC). pp. 1560–1568. IEEE (2013)
22. Pavón, J., Gómez-Sanz, J.J., Fuentes, R.: The INGENIAS methodology and tools. In: Agent-oriented methodologies, pp. 236–276. IGI Global (2005)
23. Pohl, K., Böckle, G., Van Der Linden, F.: Software product line engineering, vol. 10. Springer (2005)
24. Ramanath, A.M., Gilbert, N.: The design of participatory agent-based social simulations. Journal of Artificial Societies and Social Simulation **7**(4) (2004)
25. Sandkuhl, K., Fill, H.G., Hoppenbrouwers, S., Krogstie, J., Leue, A., Matthes, F., Opdahl, A.L., Schwabe, G., Uludag, Ö., Winter, R.: Enterprise modelling for the masses – from elitist discipline to common practice. In: Horkoff, J., Jeusfeld, M.A., Persson, A. (eds.) The Practice of Enterprise Modeling. pp. 225–240. Springer International Publishing, Cham (2016)
26. 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)
27. Stepney, S., Polack, F.: Engineering Simulations as Scientific Instruments: A Pattern Language: With Kieran Alden, Paul S. Andrews, James L. Bown, Alastair Droop, Richard B. Greaves, Mark Read, Adam T. Sampson, Jon Timmis, Alan F.T. Winfield (01 2018). https://doi.org/10.1007/978-3-030-01938-9
28. Voinov, A., Bousquet, F.: Modelling with stakeholders. Environmental Modelling & Software **25**(11), 1268–1281 (2010). https://doi.org/https://doi.org/10.1016/j.envsoft.2010.03.007, https://www.sciencedirect.com/science/article/pii/S1364815210000538, thematic Issue - Modelling with Stakeholders
29. Völter, M., Benz, S., Dietrich, C., Engelmann, B., Helander, M., Kats, L.C.L., Visser, E., Wachsmuth, G.: DSL Engineering - Designing, Implementing and Using Domain-Specific Languages. dslbook.org (2013), http://www.dslbook.org
30. Zschaler, S., Polack, F.A.C.: A family of languages for trustworthy agent-based simulation. In: Proceedings of the 13th ACM SIGPLAN International Conference on Software Language Engineering. p. 16–21. SLE 2020, Association for Computing Machinery, New York, NY, USA (2020). https://doi.org/10.1145/3426425.3426929, https://doi.org/10.1145/3426425.3426929