Plausibly Realistic Sociotechnical Simulation with Aspect Orientation

Tom Wallis

Submitted in fulfilment of the requirements for the Degree of Doctor of Philosophy

School of Computing Science

College of Science and Engineering

University of Glasgow

Table of Contents

Todo list

Chapter 1

Introduction

Some good things here

also some text

1.1 The first section header

this thesis has many features, such as content and citations, such as [1]

Chapter 2

Relevant Literature

The work composing this thesis revolves around the combination of simulation and aspect orientation. As discussed, some research on the topic was also done prior to this body of work I assume we discuss PDSF existing in the intro . There are therefore several things to discuss: work that was done on this project prior to this thesis, related work by others, and opportunities found in this related work for which PyDySoFu is well-suited. This chapter is therefore structured as follows:

section 2.1 The earlier work contextualising this thesis, summarised in [2]

section 2.2 Similar software libraries which form a body of related work to the tool described in [2]

section 2.3 and section 2.4 Aspect orientation as applied in simulation and modelling

section 2.5 Variable behaviour in simulation and modelling, a key strength of [3]

section 2.6 Research opportunities identified in these various bodies of work

2.1 Early work on PyDySoFu

PyDySoFu[1] is a Python library [3] built for making changes to the source code of a Python function as it is called, and before it is executed, while the original function definition remains oblivious to the changes being made. It was originally developed as an honours-level dissertation, which was built upon and detailled in a subsequent paper [2]. This thesis furthers that original work. To be clear

[1]Or "PDSF" for short.

about the work this thesis contains, the state of the project *before* this work began is briefly discussed here.

2.1.1 PyDySoFu's implementation and features

The original version of PyDySoFu[2] patched Python classes with additional handlers. Attributes of Python objects are usually retrieved using dot notation (i.e. `object_id.attr_id`), which evaluates internally to a call to `object_class.__getattribute__(``attr_id'')`. PyDySoFu replaces a class' built-in `__getattribute__()` method with a new one, which calls the original to acquire the required attribute.

In the case where the required attribute is not callable, the value is returned as normal. Callable attributes are modified, however. In this case, the replacement `__getattribute__()` also checks for a set of manipulations to make to the original code. These can be applied before or after the original code is run, as well as around it. A new function is returned containing a reference to the originally sought attribute, but which will search for these additional pieces of work before executing it, and can execute this work before or after the call (or both). These pieces of work are referred to as "advice", adopting aspect orientation terminology.

As discussed further in section 2.1.2, this approach is effectively an implementation of a traditional aspect orientation framework. However, unlike existing frameworks, PyDySoFu also supports a special kind of "around" advice: before a function is called, it can be rewritten. This is done by applying "before" advice which retrieves the abstract syntax tree of the target callable attribute using Python's `inspect` module (its built-in reflection), applying arbitrary transformations to the tree, and recompiling it into a Python `code` object (its representation of its internal bytecode). At this point, many things are possible: the transformation can be cached for later use, can replace the original callable's `code` object to make the transformation persistent, or can be discarded after use. This transformed code is run in lieu of the original, effectively enabling aspect orientation which can make adaptations *inside* a procedure as well as before and after its execution.

This approach also had some limitations:

[2]Further improvements have been made through this research which improve on the design, but this section is to discuss the state of the project before this work began, and the general principles around it, which remain unchanged.

- Traditional pointcuts cannot target points inside a procedure, meaning that an aspect applied "inside" its target must manage the points where its transformation is applied manually.

- Importantly, a callable object's internal bytecode cannot be replaced in Python3, leading to a rewrite discussed in Make a crossreference to the discussion of PDSF's rewrite.

- This method is significantly slower than other aspect orientation approaches, as rewriting a class' `__getattribute__` method means that *every* resolution of an object's attributes — whether they are methods or values, and including a class' built-in "magic" methods — incurs an overhead from the replaced `__getattribute__` implementation. However slight this overhead can be made, affecting Python's built-in methods on classes means that rewriting the `__getattribute__` method is unavoidably expensive due to the scale of these methods' use.

However, the goal of the original research was to develop a flexible "proof-of-concept" of aspect orientation adapting procedure definition at runtime, which was successfully achieved[2], [4].

2.1.2 Aspect Orientation & PyDySoFu

The goals of "changing a function's behaviour" and maintaining "obliviousness" in the original definition of that function speak to the goals of the aspect oriented programming paradigm[5]. Quoting their original definitions:

> "Components are properties of a system, for which the implementation can be cleanly encapsulated in a generalized procedure. Aspects are properties for which the implementation cannot be cleanly encapsulated in a generalized procedure. Aspects and cross-cut components cross-cut each other in a system's implementation. […] The key difference between AOP and other approaches is that AOP provides component and aspect languages with different abstraction and composition mechanisms."

Generally, aspect orientation is percieved to be a technique for separation of concerns. Any cross-cutting concerns can be separated from their components into aspects applied where that concern arises. The strength of aspect orientation lies in its compositional nature: developers can write short, maintainable implementations of a procedure's core purpose (for example, business logic) and ancillary concerns such as logging or security can be woven into this implementation as preprocessing, compilation, or at runtime. This compositional nature is what gives rise to aspect orientation's "obliviousness", as the procedure targetted by a piece of advice is written without regard to that fact.

The original PyDySoFu implementation was an aspect orientation library focusing on separating a function's definition from *potential changes to it*. This was used to model "contingent behaviour" — behaviour sensitive to some condition — as an original, "idealised" definition of that behaviour, plus some possible alterations. These changes might apply to many different behaviours in the same manner, and therefore represent concerns which separate cleanly into an aspect. An example would be the behaviour of a worker whose job requires focus on allocated tasks. A lack of focus could be represented as steps of the worker's tasks being executed in duplicate, out-of-order, or skipped. Assuming aspects as described by Kiczales, Lamping, Mendhekar, *et al.* are able to edit the definition or execution of a procedure[3], such contingent behaviours are well modelled as aspects.

To achieve this, a model was presented in [2] wherein aspects were developed which could change function *definitions* on each invocation of that function, contingent on program state. This allowed behavioural adaptation to be simulated in an aspect-oriented fashion. In addition, a library of behavioural adaptations called FUZZI-MOSSCITECITECITE was developed which implemented many cross-cutting, contingent behaviours in procedural simulations of sociotechnical systems.

One important contribution of this work is that PDSF aspects are effectively able to operate *inside* a target. In typical aspect orientation frameworks such as AspectJ[6], aspects operate by effectively prepending or appending work to a target, referred to as "before" or "after" pointcuts respectively. To do both is referred to as "around". By manipulating procedures within Python directly, PDSF is able to manipulate its target from a new perspective, adding (or removing) work during the target's execution[4]. Moreover, because weaving is performed dynamically, every execution of a function may perform different operations.

2.1.3 Opportunities presented by PDSF

PDSF presented several oppportunities for future research. Some salient properties of the original work include:

- It provided an aspect orientation library which could weave and unweave aspects during program execution, without relying on anything other than Python's built-in language features

[3] As opposed to simply wrapping it with additional behaviour before and/or after execution
[4] Similarly to **??**, but in an aspect oriented manner.

→ Write more PDSF opportunities

Do we need a brief explainer of what aspect orientation is here? Or will this go in the introduction? already a little in the earlier two subsections.

2.2 Dynamism in AOP

Aspect orientation frameworks have supported "dynamic behaviour" in different ways for a long time. This is largely through a technique referred to as dynamic– or runtime–weaving.

2.2.1 Dynamic and static weaving

Dynamic weaving integrates advice into a target program during its execution, as opposed to during compilation or a pre-processing step. The advantage of this is flexibility: dynamic aspect-oriented approaches have been proposed for deploying hotfixes in safety-critical scenarios where software systems cannot be taken offline to apply patchesCITECITECITE , and in adaptive mobile scenarios where software may need to alter its properties in response to its environment[7], or when debugging code to apply potential patches without reloading an entire software system[8].

To meet these needs, software systems need to check for available aspects to weave at any join point, as it is always possible that the set of applied advice has changed since the program last encountered this point. The technique therefore presents a tradeoff compared to traditional (static) aspect weaving, as illustrated in [9]. Chitchyan and Sommerville generalise this tradeoff by describing different mechanisms used to implement aspect orientation into three main categories[5], each with their own strengths:

"Total hook weaving" alters all join points where advice may be applied before runtime, so that during execution each join point "watches" for applied advice. The benefit of this approach is that aspects can be applied at any point at runtime, but this flexibility is bought at the cost of maximum overhead: at all points where weaving *may* be possible, checks for applied advice must be made.

[5]Drawing from [8], [10] where "PROSE", a particularly influential dynamic aspect orientation library, is detailled.

"Actual hook weaving" weaves hooks only to join points that are expected to be in use. This limits overhead from watching for applied advice, at the cost of flexibility: during program execution, advice may be applied or retracted *only at specific points within the system*.

"Collected weaving" weaves aspects directly into code at compilation / preprocessingsurely this isn't dynamic, Ian...?! , so as to collect advice and target codebase into a single unit. This provides exactly the necessary amount of overhead, and in many cases may result in requiring no "watching" for applied advice at all, but this limits a developer's ability to amend advice supplied at runtime.

There is an almost direct tradeoff between the number of potential join points actively checking for applied advice at runtime, and the overhead of dynamism in any aspect oriented framework, with "total hook weaving" providing complete adaptability at the expense of checking at all possible points whether advice is applied.

Another tradeoff could be seen to be the clarity of dynamically woven aspect oriented code. Aspect orientation is already criticised for the lack of clarity as to what woven code will *do* when run, and where weaving can change during program execution, static tools are less useful in making these predictions. Some tools have been produced which do provide tooling for achieving understanding as to what dynamically woven code will do when executed (also called an "Aspect Monitor", as discussed in [8]), but they are often limited or missing from a dynamically weaving framework's implementation (such as [11]). Find more citations for both dynamic weavers with aspect monitors and without. Nanning aspects? Nu?

2.2.2 PROSE

One implementation of dynamic weaving is PROSE[8], [10], a library which achieves dynamic weaving by use of a Just-In-Time compiler for Java. The authors saw aspect orientation as a solution to software's increasing need for adaptivity: mobile devices, for example, could enable a required feature by applying an aspect as a kind of "hotfix", thereby adapting over time to a user's needs. Other uses of dynamic aspect orientation they identify are in the process of software development: as aspects are applied to a compiled, live product, the join points being used can be inspected by a developer to see whether the pointcut used is correct. If not, a developer could use dynamic weaving to remove a mis-applied aspect, rewrite the pointcut, and weave again without recompiling and relaunching their project.

Indeed, the conclusion Popovici, Alonso, and Gross provide in [10] indicates that the performance issues generalised by Chitchyan and Sommerville in [9] may prevent dynamic aspect orientation from being useful in production software, but that it presented opportunities in a prototyping or debugging context.

PROSE explores dynamic weaving as it could apply in a development context, but the authors do not appear to have investigated dynamic weaving as it could apply to simulation contexts, or others where software making use of aspects does not constitute a *product*.

2.2.3 Handi-Wrap

Handi-Wrap[11] is a Java library allowing for dynamic weaving via a third-party language designed for metaprogramming, called MayaDo I want a citation for this? Probably not, but worth revisiting. . At the time of development Handi-Wrap's dynamic aspect weaving feature was novel: the aspect orientation library of note, AspectJ, wove only statically[6], and Handi-Wrap's purpose was to show that DSLs for metaprogramming could pave a way to dynamic weaving.

Baker and Hsieh implemented an aspect orientation framework which is reasonably performant, weaves dynamically, and allows for aspect orientation features to be implemented natively for greater control as compared to Handi-Wrap's then competitor, AspectJ. As a tool, Handi-Wrap demonstrated a promising approach to dynamic weaving, but the project appears to have enjoyed less attention than similar work (such as PROSE, described in **??**).

The technique used to implement Handi-Wrap (implementation via a metaprogramming-specific DSL, Maya) is familiar, in that it shares a perspective on dynamic weaving with early PyDySoFu work. The fuzzers used in [2] applied transformations to abstract syntax trees, not unlike a LISP-style macro. To quote [12] by way of contrast: "`Maya generalizes macro systems by treating grammar productions as generic functions…`".Revisit this inline quote format The two approaches have clear differences. Most notably, PyDySoFu's entire implementation *and use* is performed in Python directly, and Maya's intended purpose is metaprogramming in a more general sense.

[6] AspectJ now supports what it calls "load-time weaving" — that is, weaving aspects as classes are loaded into the JVM — but not weaving to things that are *already* loaded, meaning AspectJ still allows for only a particular flavour of dynamic behaviour.

It is possible that, while Maya provided a useful foundation to explore the dynamic weaving of aspects, its lack of adoption as a language limited handi-wrap's reach; nevertheless, it is encouraging to see another use of metaprogramming for weaving aspects at runtime.

2.2.4 Nu

Some extra things here about Nu, such as [13].

2.2.5 Binary Component Adaptation

Binary Component Adaptation[14] (BCA) is a technique for performing adaptations on software components after compilation. Though it works on already-compiled code it does provide dynamic behaviour: the technique can adapt software components via rewriting before or during the loading of its target. Like some aspect orientation techniqueswhich?! , BCA adapts a Java class loader to make its adaptations, but unlike aspect oriented approaches it does not require access to the original source of the software. For scientific simulation purposes, it could therefore be appealing in situations where adaptations are made to another researcher's simulations — assuming the original source code is not published — or in security settings investigating trust in compilers and runtimes[15]. In the present context of developing sociotechnical simulations however, this does not appear to be an advantage, particularly at a time when the source code of software components of research projects are increasingly published.

An important distinction to be made is that BCA provides an example of runtime adaptation, but does not enable an aspect oriented approach and is not developed with separation of concerns in mind. It is presented here as a useful contrast to PyDySoFu: it demonstrates an alternative technique for achieving dynamic runtime source manipulation, even if the lack of separation of concerns means it would not be well applied for this thesis.

2.3 Aspect Orientation & Modelling

Having discussed aspect orientation as it is used in a simulation context, it is natural to investigate its use in modelling research, too.

Simulation and modelling are similar topics and are often combined into a single study. However, their goals differ. Simulation typically involves the study of processes or behaviour: there is an expectation that simulations are *executed* or *run*. This often produces data. The intent of modelling is more structural in nature: models are typically observed or analysed to gain insights. Quoting Maria's introduction [16]:

> "Modeling *[sic]* is the process of producing a model; a model is a representation of the construction and working of some system of interest. A model is similar to but simpler than the system it represents.
> […]
> A simulation of a system is the operation of a model of the system. The model can be reconfigured and experimented with; usually, this is impossible, too expensive or impractical to do in the system it represents. The operation of the model can be studied, and hence, properties concerning the behavior of the actual system or its subsystem can be inferred."

Maria's definition implies that to simulate is to operate a model. Whether this model is constructed for the purpose of simulation or for study in its own right, a simplified representation of the system being studied is implicitly required for any simulation. However, modelling does not imply simulation. Models can be studied for their own merits, and many modelling frameworks exist which are made explicitly for their own study, without regard to their use in simulation[7]. Aspect orientation has seen some study in modelling, particularly for sociotechnical modelling, and while aspect-oriented sociotechnical modelling is not generally researched with subsequent simulation in mind, an important body of work is still present, and therefore important to discuss.

2.3.1 Aspect Orientation in Business Process Modelling

Aspect orientation for sociotechnical systems is particularly well studied in the business process modelling community[19], [20] find more citations for AOBPM

2.3.2 MAML & SWARM

Is MAML/SWARM really modelling, or simulation? Simulation, right?

[7] Consider UML, a well-studied modelling framework which is generally not used for any kind of simulation — depending on its use, it often cannot be — and for which many alternatives now exist specifically to address this limitation [17], [18].surely one more here to round this out.

2.3.3

2.4 Aspect Orientation & Simulation

The simulation section *badly* needs revisiting.

Surprisingly, little literature exists pertaining specifically to the use of aspect-orientation in a simulation context. Aspect orientation is often applied to modelling as discussed in section 2.3, used to compose a perspective of the world from individual parts, but in a way which isn't necessarily executable or able to produce data.

Early in the history of aspect orientation as an emerging paradigm, there was some interest in its use for scientific simulation. [21] discuss that computer simulations require code for both observation of a simulation and the simulation itself, and that misuse of this could cause what is in effect a kind of Hawthorne Effectdoes hawthorne effect need a citation? , where the inclusion of observation code intertwined with simulation code might influence the outcome of an experiment. They suggest that improving simulation technologies could combat this approach. Aspect Orientation, being developed specifically with obliviousness in mind, is an ideal candidate which Gulyás and Kozsik identify.

Much of the literature concerning aspect-oriented programming and simulation focus on tooling support for aspect-oriented simulation, rather than investigations into its efficacy. For example, attempts have been made to integrate aspect orientation into new tools [22] throw more in here from excel , or into existing ones [23]throw more in here from excel . Rethink this. The angle is, "much of the lit strays from any kind of real-world testing of sim tech — why?"

Some experiments specifically using aspect orientation in the implementation of process-based simulations also exist[24] include more! . For example, Ionescu, Piater, Scheuermann, *et al.* apply aspect orientation in a nuclear disaster prevention simulation. Their motivation is that code can become complex to maintain over time and changes to the scientific zeitgeist or to regulatory requirements become costly as technical debt mounts. Aspect orientation therefore allows developers to separate functionality into distinct modules more easily, without disturbing the underlying codebase.

2.4.1 Aspect-oriented L-Systems

Aspect-orientation is also applied in other simulation paradigms. Cieslak, Seleznyova, Prusinkiewicz, *et al.* investigated the use of aspect orientation in L-system based simulations [25]. An L-system[26] is defined by a set of symbols, an initial string composed of these symbols, and a set of rules for rewriting substrings. While being a powerful tool for representing fractal structures, they were originally concieved of for plant modelling (and still see the most use in this field).

Cieslak, Seleznyova, Prusinkiewicz, *et al.* note that some details of plant modelling are actually cross-cutting concerns against many plants or families of plants. To represent these, they introduce a new language to describe plant models which makes use of aspect orientation to represent these cross-cutting concerns. They test the approach by representing carbon dynamics, apical dominance and biomechanics as cross-cutting concerns that are integrated into a previously published model of kiwifruit shoot development. Cieslak, Seleznyova, Prusinkiewicz, *et al.* hope that these cross-cutting concerns might work in other models too, but this is untested. The use of an aspect in a new model, when developed for another, seems untested in the community's literature writ large and is a noted omission in the conclusion of this particular work.

2.5 Variable Behaviour in Simulation and Modelling

In [2], PyDySoFu was used to model behaviour that changed as the simulation progressed. Behaviour undergoing variance appears in literature from many fields, but some themes stand out. Researchers are often interested in:

→ Removing small variations from datasets in order to mine the original process (that real-world actors might be deviating from), referred to as sanitisation,

→ Inserting variations so as to produce datasets with

2.5.1 Process mining & variation in behaviour

Is this going from data to a model? Models to data? Potentially multiple models either way depending on their use? Consider this and possibly restructure.

2.5.2 Process mining & variation in data

Write a short subsection on the trend of santising data / coping with noisy datasets. Sometimes behavioural variance isn't desirable. Consider making noisy data a subsubsection of section 2.5.3.

2.5.3 Synthetic datasets exhibiting variance

2.6 Research Opportunities in the Literature

One notable omission from the set of research themes outlined in section 2.5 is that variations on processes aren't well studied in their own right. That is to say, behavioural variation is typically treated as a nuisance to the researcher or practicioner interested in a model or dataset, and the variations and their impact on simulations are not studied on their own. Some work existscite work that focuses on variations specifically , but the majority of this is done in the context of tooling, i.e. the representation of variation for their use in another research context, where they are not the subject.

As an aspect orientation framework capable of runtime adaptation of a target system which can manipulate a join point from inside what are our research opportunities given the above...?

Bibliography

[1] W. Wallis, W. Kavanagh, A. Miller, and T. Storer, "Designing a mobile game to generate player data-lessons learned," 2020.

[2] T. Wallis and T. Storer, "Modelling realistic user behaviour in information systems simulations as fuzzing aspects," in *International Conference on Advanced Information Systems Engineering*, Springer, 2018, pp. 254–268.

[3] ——, *Pydysofu*, `https://github.com/twsswt/pydysofu`, 2018.

[4] ——, "Process fuzzing as an approach to genetic programming," in *Proceedings of the SICSA Workshop on Reasoning, Learning and Explainability*, K. Martin, N. Wiratunga, and L. S. Smith, Eds., ser. CEUR Workshop Proceedings, vol. 2151, Aberdeen, UK: CEUR-WS.org, Jun. 2018. [Online]. Available: `http://ceur-ws.org/Vol-2151/Paper_S3.pdf`.

[5] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.-M. Loingtier, and J. Irwin, "Aspect-oriented programming," in *European conference on object-oriented programming*, Springer, 1997, pp. 220–242.

[6] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, and W. Griswold, "An overview of aspectj," in *ECOOP*, 2001.

[7] J. C. Hveding, "An aspect-oriented approach to adaptive systems," M.S. thesis, Institutt for datateknikk og informasjonsvitenskap, 2005.

[8] A. Popovici, T. Gross, and G. Alonso, "Dynamic weaving for aspect-oriented programming," in *Proceedings of the 1st International Conference on Aspect-Oriented Software Development*, ser. AOSD '02, Enschede, The Netherlands: Association for Computing Machinery, 2002, pp. 141–147,

ISBN: 158113469X. DOI: 10.1145/508386.508404. [Online]. Available: https://doi.org/10.1145/508386.508404.

[9] R. Chitchyan and I. Sommerville, "Comparing dynamic ao systems," 2004.

[10] A. Popovici, G. Alonso, and T. Gross, "Just-in-time aspects: Efficient dynamic weaving for java," in *Proceedings of the 2nd International Conference on Aspect-Oriented Software Development*, Association for Computing Machinery, 2003, pp. 100–109.

[11] J. Baker and W. Hsieh, "Runtime aspect weaving through metaprogramming," in *Proceedings of the 1st international conference on Aspect-oriented software development - AOSD '02*, ACM Press, 2002. DOI: 10.1145/508386.508396. [Online]. Available: https://doi.org/10.1145%2F508386.508396.

[12] J. Baker and W. C. Hsieh, "Maya: Multiple-dispatch syntax extension in java," in *Proceedings of the ACM SIGPLAN 2002 Conference on Programming Language Design and Implementation*, ser. PLDI '02, Berlin, Germany: Association for Computing Machinery, 2002, pp. 270–281, ISBN: 1581134630. DOI: 10.1145/512529.512562. [Online]. Available: https://doi.org/10.1145/512529.512562.

[13] R. Dyer and H. Rajan, "Supporting dynamic aspect-oriented features," *ACM Transactions on Software Engineering and Methodology*, vol. 20, no. 2, pp. 1–34, Aug. 2010. DOI: 10.1145/1824760.1824764. [Online]. Available: https://doi.org/10.1145%2F1824760.1824764.

[14] R. Keller and U. Hölzle, "Binary component adaptation," in *European Conference on Object-Oriented Programming*, Springer, 1998, pp. 307–329.

[15] K. Thompson, "Reflections on trusting trust," in *ACM Turing award lectures*, 2007, p. 1983.

[16] A. Maria, "Introduction to modeling and simulation," in *Proceedings of the 29th Conference on Winter Simulation*, ser. WSC '97, Atlanta, Georgia, USA: IEEE Computer Society, 1997, pp. 7–13, ISBN: 078034278X. DOI: 10.1145/268437.268440. [Online]. Available: https://doi.org/10.1145/268437.268440.

[17] D. DORI, "Object-process Analysis: Maintaining the Balance Between System Structure and Behaviour," *Journal of Logic and Computation*, vol. 5, no. 2, pp. 227–249, Apr. 1995, ISSN: 0955-792X. DOI: 10.1093/logcom/5.2.227. eprint: https://academic.oup.com/logcom/

article-pdf/5/2/227/6244720/5-2-227.pdf. [Online]. Available: https://doi.org/10.1093/logcom/5.2.227.

[18] A. A. Mitsyuk, I. S. Shugurov, A. A. Kalenkova, and W. M. van der Aalst, "Generating event logs for high-level process models," *Simulation Modelling Practice and Theory*, vol. 74, pp. 1–16, May 2017. DOI: 10.1016/j.simpat.2017.01.003. [Online]. Available: https://doi.org/10.1016%2Fj.simpat.2017.01.003.

[19] I. Machado, R. Bonifácio, V. Alves, L. Turnes, and G. Machado, "Managing variability in business processes," in *Proceedings of the 2011 international workshop on Early aspects - EA '11*, ACM Press, 2011. DOI: 10.1145/1960502.1960508. [Online]. Available: https://doi.org/10.1145%2F1960502.1960508.

[20] C. Cappelli, J. C. Leite, T. Batista, and L. Silva, "An aspect-oriented approach to business process modeling," in *Proceedings of the 15th workshop on Early aspects - EA '09*, ACM Press, 2009. DOI: 10.1145/1509825.1509828. [Online]. Available: https://doi.org/10.1145%2F1509825.1509828.

[21] L. Gulyás and T. Kozsik, "The use of aspect-oriented programming in scientific simulations," in *Proceedings of Sixth Fenno-Ugric Symposium on Software Technology, Estonia*, 1999.

[22] A. Aksu, F. Belet, and B. Zdemir, "Developing aspects for a discrete event simulation system," in *Proceedings of the 3rd Turkish Aspect-Oriented Software Development Workshop*, Bilkent University, 2008, pp. 84–93.

[23] M. Chibani, B. Belattar, and A. Bourouis, "Using aop in discrete event simulation: A case study with japrosim," *International Journal of Applied Mathematics, Computational Science and Systems Engineering*, vol. 1, 2019.

[24] T. B. Ionescu, A. Piater, W. Scheuermann, E. Laurien, and A. Iosup, "An aspect-oriented approach for disaster prevention simulation workflows on supercomputers, clusters, and grids," in *2009 13th IEEE/ACM International Symposium on Distributed Simulation and Real Time Applications*, IEEE, 2009. DOI: 10.1109/ds-rt.2009.35. [Online]. Available: https://doi.org/10.1109%2Fds-rt.2009.35.

[25] M. Cieslak, A. N. Seleznyova, P. Prusinkiewicz, and J. Hanan, "Towards aspect-oriented functional–structural plant modelling," *Annals of Botany*, vol. 108, no. 6, pp. 1025–1041, Jul. 2011.

DOI: 10.1093/aob/mcr121. [Online]. Available: https://doi.org/10.1093%2Faob%2Fmcr121.

[26] A. Lindenmayer, "Mathematical models for cellular interactions in development i. filaments with one-sided inputs," *Journal of theoretical biology*, vol. 18, no. 3, pp. 280–299, 1968.