Social simulation comparison in arbitrary problem domains: first steps towards a more principled approach

Stuart Rossiter[1], Jason Noble[1], and Keith R.W. Bell[2]

[1] University of Southampton, UK
sr1@ecs.soton.ac.uk, jn2@ecs.soton.ac.uk
[2] University of Strathclyde, UK
keith.bell@eee.strath.ac.uk

Abstract. We outline a simulation development process, backed by a software framework, which focuses on developing and using a *partial* conceptual model as a 'lens' to compare and possibly re-implement existing models in a chosen problem domain (as well as to design new models). To make this feasible for existing models of arbitrary structure and background social theory, we construct our (partial) conceptual model in a way that acknowledges that it is a base representation which any individual model will typically add detail to, *and* abstract away from, in various ways which we argue can be formalised. A given model's design is fitted to the conceptual model to capture how its structural architecture (and selected aspects of the system's state and driving processes) map to the conceptual model. This fit can be used to produce incomplete skeleton code which can then be extended to produce a simulation. Along the way, we use robust decision-making to provide a useful frame and discuss how our approach differs from others. This is inevitably a preliminary approach to a broad and difficult problem, which we explore in the conclusions.

Keywords: modelling process, model comparison, M2M analysis, software framework, model-driven-design, robust decision-making, LTPA

1 Motivation

Model comparison is a key process needed to help mature social simulation as a discipline, and encompasses related interests in model reuse, alignment and replication [3,2,7,20]. It is also highly relevant for the use of social simulations in long-term policy analysis (LTPA). The robust decision-making approach [11,12] has focused on interactive scenario exploration, as part of a process whereby policy options can be developed which are *robust* to as wide a range of plausible scenarios as possible.[3] The scenarios are produced by a **scenario generator**, which could be a single model (varied parametrically to produce scenarios)

[3] The authors also term it Computer-Assisted Reasoning (CAR), but we feel this is too general a term to be useful, and prefer 'robust decision-making'.

or some ensemble of models.[4] The inherent deep uncertainty in social systems means that attempting to assign 'likelihoods' to each scenario is inappropriate, and thus the generator aims only for plausibility and variety of scenarios. (The devil is, of course, in the detail of how one assesses plausibility, and when a policy is 'robust enough'.)

In our view, the scenario generator should really be some ensemble of models that represents *theoretical plurality*, not just parametric variation of a single model or family of related models. This is vitally important because, even if we ignore (or refute) the deep uncertainty of socal systems, the reality is that there is a lack of consensus on valid theory, and the predictive accuracy of social simulation is extremely limited, particularly across longer timeframes. (Moss [15] takes a strong stance on this.) Thus, although time and funding often precludes it, a scientifically honest approach would include simulations covering a wide range of theoretical alternatives. In concrete terms, this means that *variation in model structure is equally, or more, important than the variation of parameters (and the related focus on the accuracy of any empirical data used).* Robust decision-making has focused much more on the scenario exploration tools and process than on how to construct a good scenario generator. Their extended example for sustainable development [12] uses a single system dynamics (SD) model. Bankes [4] used an ensemble of models, but using bootstrap resampling of training data to create a family of neural net models (i.e., although the differing neural nets created different model 'structures', there is still a single overarching theoretical backdrop).

One of the reasons why we are interested in this robust-decision-making frame is that it also represents a reasonable analogue for the field of social simulation in general. For a given problem domain, models in the literature that investigate it can be identified (assuming we can decide what is 'close enough' to the chosen domain to count). This set of models effectively acts as a scenario generator, producing a variety of possible future scenarios. Our aim as scientists is ideally to be able to compare all these models, and advance our understanding of the domain by adding, extending and combining theoretical elements of them (as well as validating and replicating their results).[5] However, this is made very difficult by aspects such as disciplinary (or paradigm-specific) silos, a multiplicity of implementation languages and frameworks, lack of access to model code, overly brief documentation in papers and, perhaps above all, radically differing theoretical or conceptual frameworks. (Different modelling paradigms tend to exist because of differences in overarching conceptual tenets, though these historical differences may be weakening as hybrid modelling becomes more common.) In contrast, this is *unlike* robust-decision-making, in that the exact 'levers' (interventions) to be investigated, and the measures by which the outcomes are judged, may

[4] Boundaries between *a model* and *models* can begin to blur where, for example, a model has parameters which act as switches to turn on and off different structural alternatives (e.g., alternative decision-making algorithms).

[5] Indeed, the ultimate aim of this understanding is often to inform policy, and thus there is the same idea of designing policy robust to a range of theories.

differ considerably amongst different pieces of research; robust-decision-making sets these up as part of the process, and the scenario generator has to conform to them.[6]

Therefore, in general, it is important to be able to compare a plurality of models for the same problem domain. However, this plurality makes model comparison more difficult: we really need ways to compare alternative model designs via some common 'lens', even when they appear to radically differ structurally. Ideally, this comparative process would also extend into the realm of model implementation, where it also facilitates code reuse.

1.1 Existing Approaches

Conceptual frameworks exist for specific problem domains which aid model assessment, understanding and comparison; e.g., Chappin & Dijkemma's for energy transitions [5], AMES for particular forms of electricity markets [22], and MR POTATOHEAD for land use models [16]. All of these provide some support for translating this conceptual model into software: what software engineering terms **model-driven design** [8, §4]. However, all these examples are tied to an agent-based paradigm, and focus more on a *minimal* representation level. This does not meet our intuitive idea of a lens as some potentially-incomplete base representation *which does not constrain how a model might be mapped to it.* That is, we would expect individual models to both add detail to, *and* abstract away from, this base representation in arbitrary ways. This is a key principle we attempt to include (inevitably in a limited fashion), and we term it **representational flexibility**.

We can also regard modelling paradigms and their software frameworks similarly (as conceptual frameworks supporting model-driven design), but where the conceptual framework is a generalised one for any system. We would like some process which does not restrict the modelling paradigms of the models compared.

Finally, there has been some research on protocols that try to standardise how models are described [18,17], which should definitely aid in model comparison. However, much of this is tied to modelling paradigm and focuses on cross-cutting aspects of models (such as the treatment of time or the representation of space) rather than how the theory-driven structure relates to the real-world system. It is also, by its nature, a textual process unrelated to model construction.

2 High-Level Approach

The overall proposed process is shown in figure 1, which we attempt to explain in what follows.

[6] They use the XLRM analysis framework. The 'relationships' (R) comprise the model, which takes into account externalities (X) and the policy levers (L) in place to produce outcomes, where we are interested in particular measures (M) of scenario desirability.

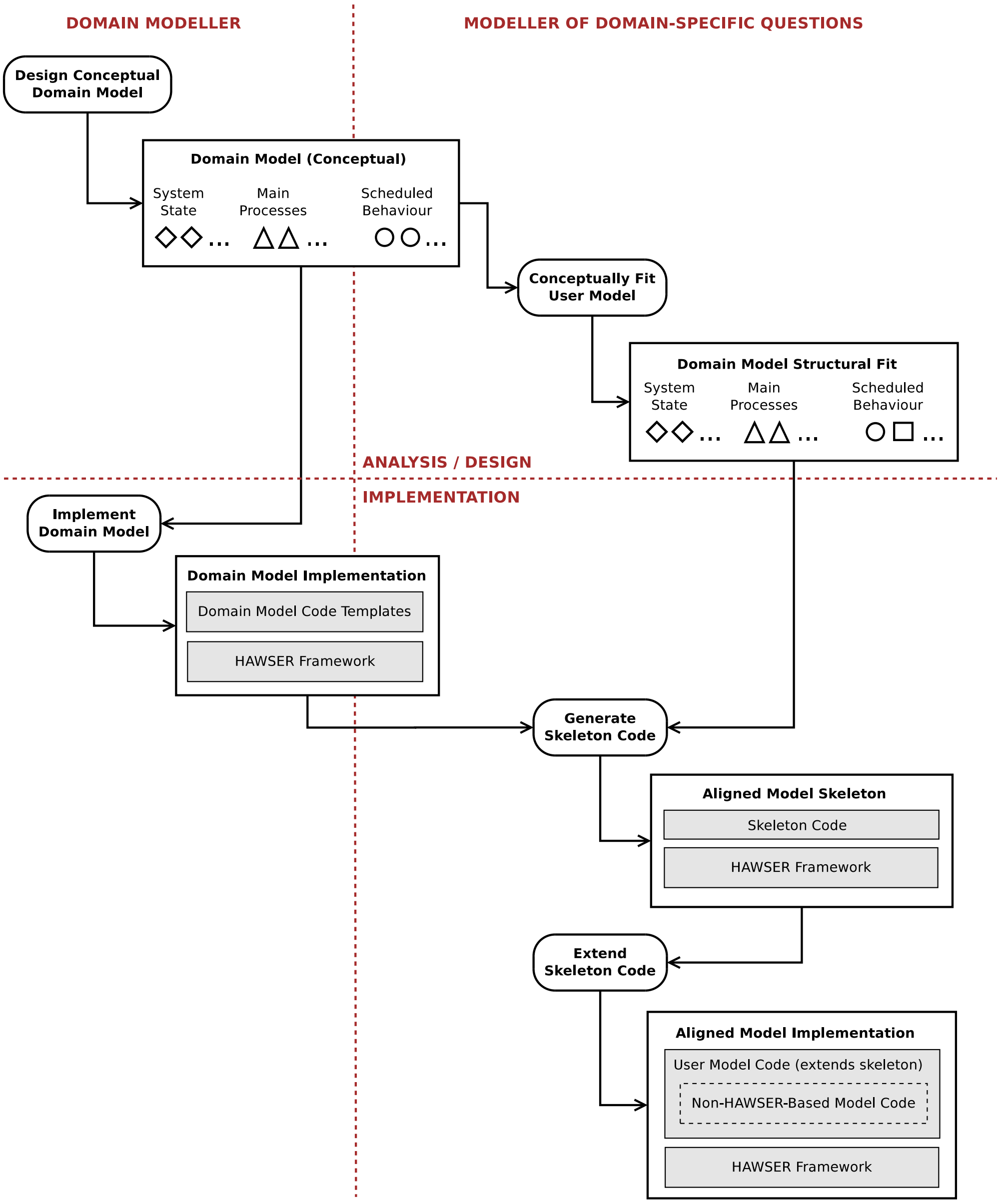

Fig. 1. A summary of the overall proposed modelling process, as a UML activity diagram with partitioning for different roles (vertical) and stages (horizontal) in the process. Shaded boxes represent code. The diamond, triangle, and circle represent instances of each of the main types of domain model element. The square in the Domain Model Structural Fit box (parameter) highlights how the behavioural fit can result in new entities, as well as existing entities.

In simulation terms, the envisaged common lens can be regarded as a *partial* conceptual model of a system, where the system's scope is determined by the problem domain: "The art of model building is knowing what to cut out, and the purpose of the model acts as the logical knife. [...] Always model a problem. Never model a system" [21, §3.5.1].[7] Unlike other model-driven approaches, it is *not* a complete model design: it focuses only on those aspects which can realistically be generalised, limiting itself to the structural 'architecture' of the system, and selected aspects of system state and driving processes. The structural architecture defines behavioural classifications, but no further behavioural details (which will necessarily differ in ways which we cannot capture in a prescribed way). Our process defines ways to build this conceptual model and fit existing models to it, incorporating the required representational flexibility.

Setting its partial nature aside, this conceptual model is almost exactly what is termed a **domain model** in software engineering [8], and we use this term henceforth (with the 'partial' implicit). Evans' book [8] does a good job of characterising the benefits that it can bring (such as a "ubiquitious language" for shared discussion).

2.1 Domain Model Structure

Intuitively, these generalisable details will often be structures that are imposed by legal, technical, political or social constraints that we do not envisage changing over simulation timescales (roughly 'medium-term': very few, if any, social simulation researchers would think it appropriate to go beyond this).[8] Such features are also ones which *presuppose as little social theory as possible*, which is what we should expect since social theory is the main mechanism by which models differ (and thus is not generalisable).[9] They are aspects that concretely exist 'out there' and would be included in an expert participant's account when no conceptually-constraining elicitation method (such as systems thinking's causal loop diagrams [21, §5]) was used. This lends itself to an **object-oriented representation**, with object classes corresponding to real-world entities, and is similar to a systems-theoretic design [23, §1.1.1]. We supplement this with **workflows** [1] to model system processes.

In practice, the approach will work best for problem domains where there are enough of these constraints for the conceptual model to be 'meaty' enough to be useful. Two good examples are (a) electricity markets, where the physics of electricity flow and the difficulty of storing it mean that electricity markets and infrastructure *have* to work in a certain way (at least unless radical disruptive

[7] Theory also drives system scope to some degree, and we explain why this is less problematic than it might seem later.

[8] That is, there will not be any changes over the course of the simulation which are fundamental and disruptive enough to change these 'structural goalposts'. This assumption is implicitly embedded in *all* simulation models, unless they are explicitly trying to model such change. (Even then, they can only cover a set of possibilities which the modeller can conceive of.)

[9] See section 4 for some discussion on the use of the word 'theory'.

technologies emerge); (b) healthcare, where the biological constraints of disease pathologies and available treatments are allied to structures of health systems that tend to define generalised care pathways. We use the example of electricity markets in what follows.

The entities whose behaviour drives the system are termed **behavioural entities**. In the real-world, this behaviour is often governed by **processes**, some of which can also be generalised in the same way as the entities. For example, because electricity cannot easily be stored, it has to be generated in real-time as it is consumed (and this demand can only be approximated beforehand). Thus, *all* electricity markets have 'balancing markets' where generators make bids and offers to increase or decrease generation as required in real-time. Equally, constructing a new power plant requires planning permission, consent from the transmission network operator, and local work to physically connect it to the grid.

In addition, there is often specific information in the real-world that is consulted by many actors to guide decision-making. This typically relates to the shared environment in which the participants are operating. In our electricity markets example, one obvious candidate is information on the topology of the electricity transmission network and what is attached to it, since that governs who can use and supply electricity, and thus the strategic options for buyers and sellers in the market. If we abstract away the *access* to this information (and aspects such as unreliable transmission of it), we can view this as **core system state** shared as information.

These distinctions (behavioural entity, process and core system state) are also made with one eye on model implementation. They corresponds to aspects we typically see in an object-oriented simulation: a set of object classes, some form of schedule to centrally drive object behaviour, and objects encapsulating system state which are often visualised and/or have statistics derived from them. The latter are typically where we derive our measures of scenario favourability from.

2.2 Analysing Models Using The Domain Model

The domain model can then be used either to design new models or, more commonly, to analyse and compare existing models. As a general scientific principle, all models should be able to explain how their theory relates to a more atheoretical understanding of the real-world system. (Indeed, many computational sociologists reject orthodox economics precisely because they feel it fails on this count [14].) Thus, they should be able to explain themselves in terms of the domain model, and the process of doing so is a valuable scientific exercise as well as being extremely useful for model comparison. We refer to this step as **structural domain model fitting** (of the individual model in question).

This does *not* mean some kind of 1–1 mapping, or that the model must only extend the basic building blocks of the domain model: it means that the model's design can be derived from the domain model via a series of extensions *and abstractions*, which could be more or less 'radical' depending on the model. (The domain model is a partial, 'atheoretical' view of the domain, not some normative design for any model of it.)

Both domain models and the fitting process have a particular form and set of design heuristics which makes this possible in a relatively formal way. Because we are trying to model entities and some of the processes which drive their interactions, we are necessarily building up a *causal-mechanism view* of the system. Thus, we should expect that models which abstract away the causal mechanism (e.g., statistical regression models) or abstract away the entity-instance aspect of reality (e.g., system dynamics) will necessarily require a more 'involved' fit: such models have to work harder to explain themselves in real-world terms. Since a regression model has the core relationship between its independent and dependent variables as an 'unknowable' statistically-fitted set of associations, the process will make that clear whilst mapping the variables to existing or new parts of the system that they relate to (see section 3.3).

This is also why our technically-incorrect assumption that only the problem domain defines the system's scope is acceptable. Although theory also drives system scope to some degree (as well as driving the *representation* of that system), representational flexibility means we are not excluding anything outside our domain model and thus theory-driven differences in scope will come out in the fitting process.

2.3 Model Implementations

Model implementations are based on a shared implementation of the domain model. The domain model fit provides ways to formally specify the main structural extensions and abstractions, which can then be used to produce *skeleton model code* from the domain model code; i.e., a partial implementation for the model which is then fleshed out by the modeller with code for the detailed behaviour of entities within the structure.

For existing models, one can either *re*-implement them in this mapped-to-the-domain-model way or extend this skeleton code to act as a 'wrapper' for the existing model (i.e., code which controls the execution of the existing model, and manipulates its inputs and outputs so as to conform to the domain model fit).[10]

Primarily due to space limitations, we focus on analysis and design in the remainder of this paper, making some passing comments on implementation aspects. (Section 4 gives a summary of what we have actually realised to date.)

2.4 Possible Usage Modes for the Process

There are different ways in which the process could be used, some of which represent grander visions than others. At the simpler scale, a domain model could be constructed just to guide the design and development process for a single model (or a closely related set of models). It may prove valuable later, unexpectedly or

[10] In software design pattern terminology, this wrapper code acts as a gateway [9, p.466].

not, in trying to compare this model with others, or in developing new models for the same domain.

In a more comparative mode, a domain model could be produced to help understand and compare a set of existing models. If done convincingly, this could influence subsequent research, with those in agreement ensuring that their model designs can be mapped to it.

Finally, in a policy mode, a domain model could be defined by, or with, policy-makers. It would be used to state the 'lens' that is expected, so that 'real world' concepts and data definitions understood by the stakeholders were used. This would then allow competing models (say from different groups of academics), which could be compared at this shared level.

3 Details via an Abstracted Example

Figures 2 and 3 provide representational summaries of the 'whole picture' for a hypothetical domain model (figure 2) and a particular model implementation which is being fitted to it (figure 3). We explain this detail in stages below, but space restrictions mean we omit many subtleties.

3.1 Core System State

The domain model provides a set of **core information entities (CIEs)**, which capture state that conceptually acts as information many actors would tend to want to consult for decision-making if it was available. (Models may or may not model the accessibility of information that exists in reality.) This state is supported by metadata and visualisations that reflect the statistics and trends which are of interest (both to system actors and to the researcher running the simulation).

If we consider the example of electrical power generation system topology discussed earlier, there are obvious useful visualisations of the network and metadata such as the total theoretical supply and demand capacities. Representational flexibility is supported in various ways: representing the system as a network of zones allows differing levels of abstraction (including a single zone where network structure is not required); configuration options allow certain levels of detail (e.g., voltage levels) to be omitted; and flexible technology definitions for power plants allow models to define their own technology classification, whilst still getting the benefits of shared visualisations and metadata.

In the domain model implementation, CIEs are a set of runnable-as-is objects. A user model implementation will configure these as required, and may extend them (perhaps with extra objects). Areas labelled A in figures 2 and 3 show this schematically.

3.2 System Processes (Scheduling)

Generalisable processes in the real-world system are represented as **workflows**, which allows the conceptual design and implementation to be closely related:

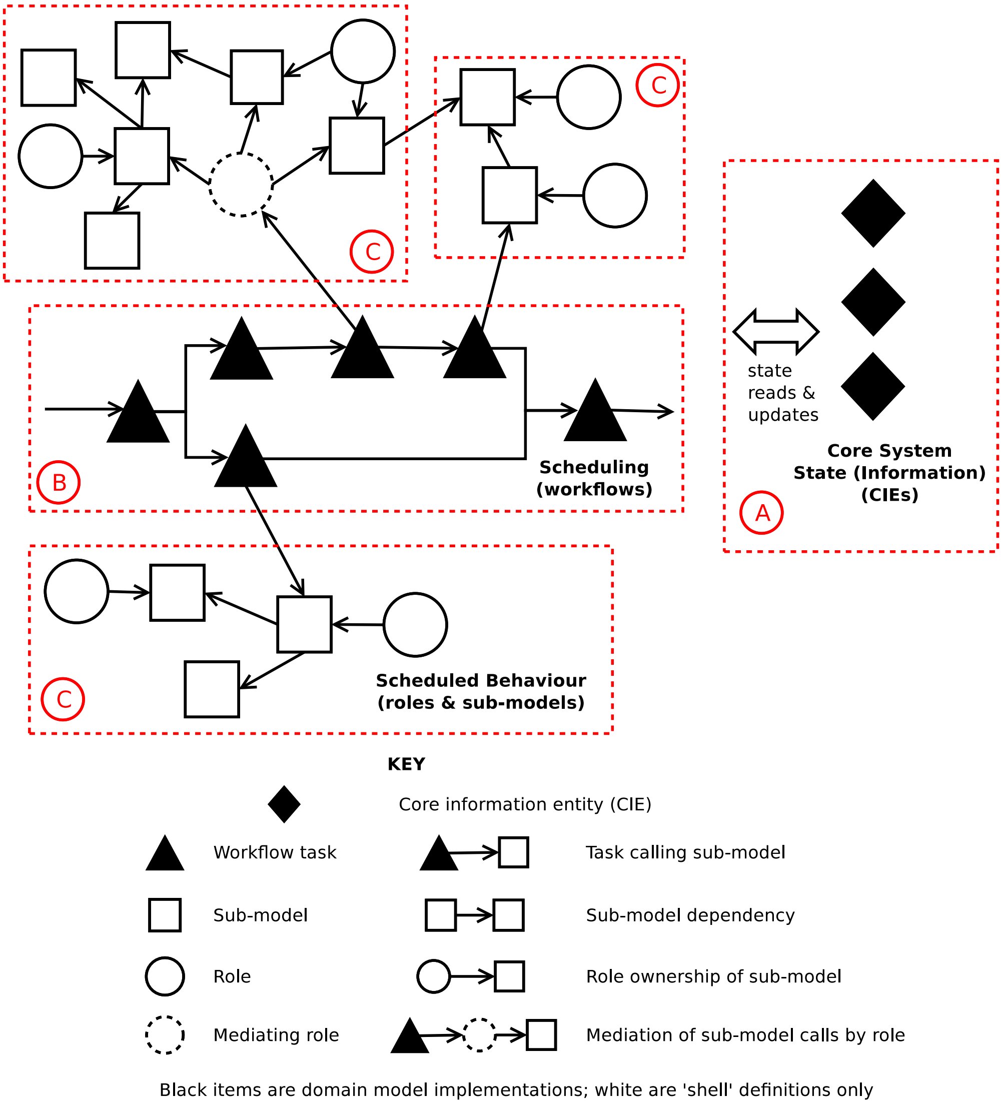

Fig. 2. An example representational domain model, showing the conceptual elements of the process. The terminology and lettered areas are explained in the main text. To avoid clutter, some of the workflow tasks are not 'expanded out'.

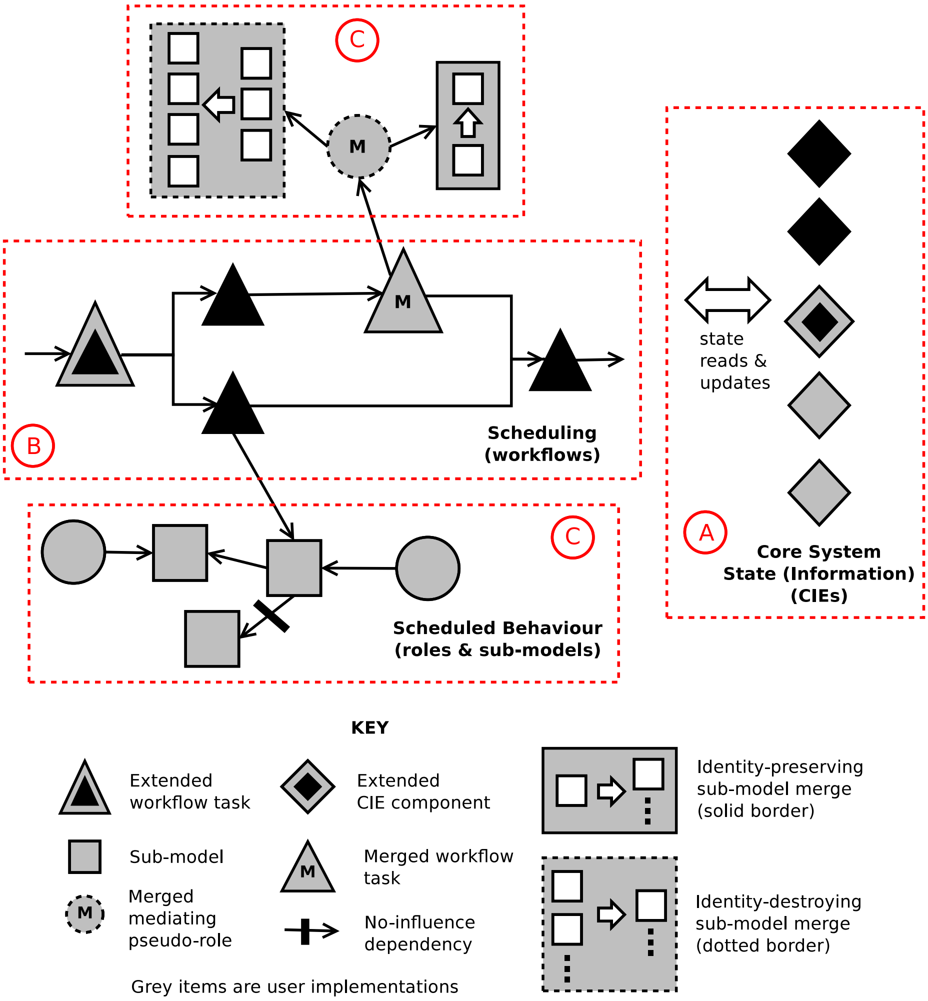

Fig. 3. An example representational model implementation, showing how one possible existing model might be fitted to the domain model of figure 2. The terminology and lettered areas are explained in the main text. The key extends that of figure 2.

workflows can be designed conceptually in the same tool used to implement them (in our case the YAWL [10] platform). Workflows provide a rich set of scheduling patterns and, crucially, can conceptually represent the parallelism that almost always exists in the real-world system.

Representational flexibility is achieved by allowing user models to **merge** workflow tasks, extend existing ones, and control the ordering of task execution (thus specifying how conceptual parallelism translates into sequential computation); there are also various related design heuristics. User models thus specify a set of workflows derived from the domain model ones; see areas labelled B in figures 2 and 3. Since merging tasks may mean that there is now no well-defined sub-model to call, special mediating roles (similar to those in section 3.3) may be required.

At the implementation level, the HAWSER framework [19] provides the bridge from these to behavioural entities, with the latter using the MASON [13] agent-based framework. This separation of concerns (behaviour and its orchestration) is a core software engineering concept [6], promoting reuse and layered design.

3.3 Scheduled Behaviour

We classify the behaviour into a set of well-defined **sub-models**, with sets of **sub-model dependencies** which define the main interactions and informational dependencies that occur in the real-world system.[11] The detail *within* these sub-models is unspecified. There are general restrictions on what sub-models can represent, which help avoid some particular conceptual difficulties; this just means that such things are part of the 'full' detail outside the scope and/or resolution of the domain model. (We do not discuss this further here.)

As is generally required in causal-mechanism-based models, we define a conceptual boundary between internal and external sub-models (e.g., Sterman [21, §3]), though this can be 'overridden' during the fitting process. Internal ones are generally more fine-grained, and are expected to provide most of the modelled interaction.[12] They are related to **roles**, which are taken up by individuals or well-defined groups of such (e.g., firms). As in the real world, implementing entities can take on multiple roles (and this provides some modelling flexibility). External sub-models have no associated roles. They may be relatable to groups of individuals (e.g., an industry sector) or environmental processes (e.g., weather).

The domain model fit will result in an adjusted set of sub-models and dependencies. Models may break down 'external' sub-models into much more detail,

[11] The dependencies specify only what interactions of information and possible action occur (i.e., that a dependency exists), not when and how these occur (i.e., not how the dependency should play out in implementation terms). Dependencies also have types, but we do not discuss that here.

[12] This does not necessarily mean that they are the main *drivers* of system-level patterns: that is a question of sensitivity analysis.

and these may interact extensively with 'internal' ones; thus, which behaviour is endogeneous and which exogeneous is still up to the model. More importantly, the fit may include new sub-models which are **merges** of others, and behaviour originally in one domain sub-model may be split amongst new ones.

Workflow tasks normally call behavioural sub-models directly. However, there are occasions when the temporal flow of actions is, by its nature, unspecifiable as a meaningful generalisation. For example, consider modelling the trading of electricity in some time period. In the real world, buyers and sellers interact via various market mechanisms, depending on the country (e.g., centralised spot/pool markets, government tenders, or decentralised bilateral contracts). We can generically model the roles and behaviours involved, but not the 'control logic' of what would drive what in a computational model. Thus, we support artificial **mediating roles**, which workflow tasks can call in such circumstances. It is then this role's responsibility to orchestrate the actions of a defined set of sub-models.[13]

Figures 2 and 3 show these ideas schematically (areas labelled C). The top-left C area in figure 2 shows a mediating role.

A More Concrete Example. Figure 4 shows an example of some explicitly named sub-models, roles and dependencies. The Net Consumer role represents net consumers of electricity.[14] Their main behaviour is to produce demand for electricity (Electricity Demand sub-model), but this is also a role where the set of instances of the role typically changes as consumers enter or leave the system; the Instance Management sub-model is a convention to represent this.[15]

In this case, Instance Management includes things such as changes in numbers of households (hence demographic factors and the dependency on social groups) and industrial consumers (hence the dependencies on supplier price schemes, connection criteria by operators, and non-electricity markets).

Figure 5 provides an illustrative example of a fit to the domain model concepts in figure 4. Let us assume the existing model has a component calculating total electricity demand for each new time step, where this is: the previous value, plus a term representing relative gas prices, plus some fitted constant; all multiplied by a value representing the number of consumers via some population projection data.

Firstly, we can still consider this as explicitly modelling a Net Consumer role, just with only a single instance (for the whole system). Secondly, the structure of the demand function cleanly separates out into a term for electricity demand (Electricity Demand sub-model) and a multiplier for the number of consumers (Instance Management sub-model). The latter uses only population projection

[13] This idea has analogues with concepts in theory, such as Adam Smith's 'invisible hand'.

[14] They may have small amounts of generation, such as a household with photovoltaic panels, but they are a consumer on balance.

[15] This convention is appropriate here because the process governing changes in consumer instances is largely independent from that governing electricity demand per consumer. If it were not, there is an alternative convention.

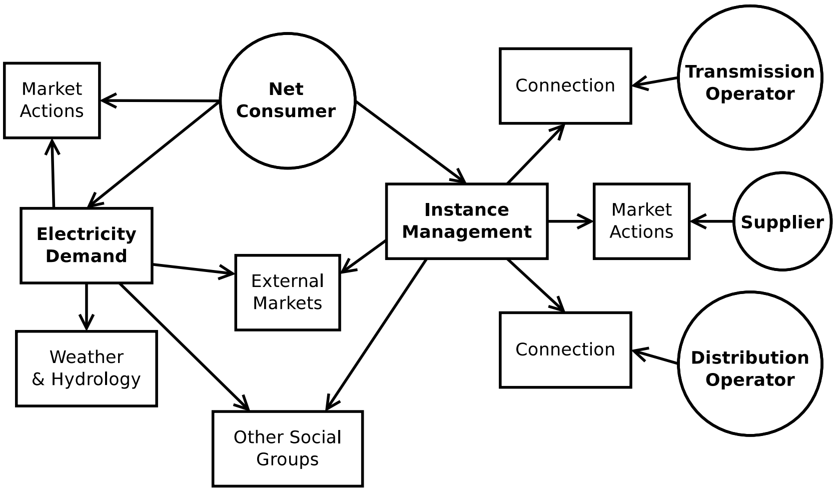

Fig. 4. Sub-model dependencies and role relationships for two sub-models of the Net Consumer role (called as part of a workflow task representing the real-time operation of the power system). The symbols used are as in figure 2. Sub-models without owning roles are external ones. Only direct dependencies are shown.

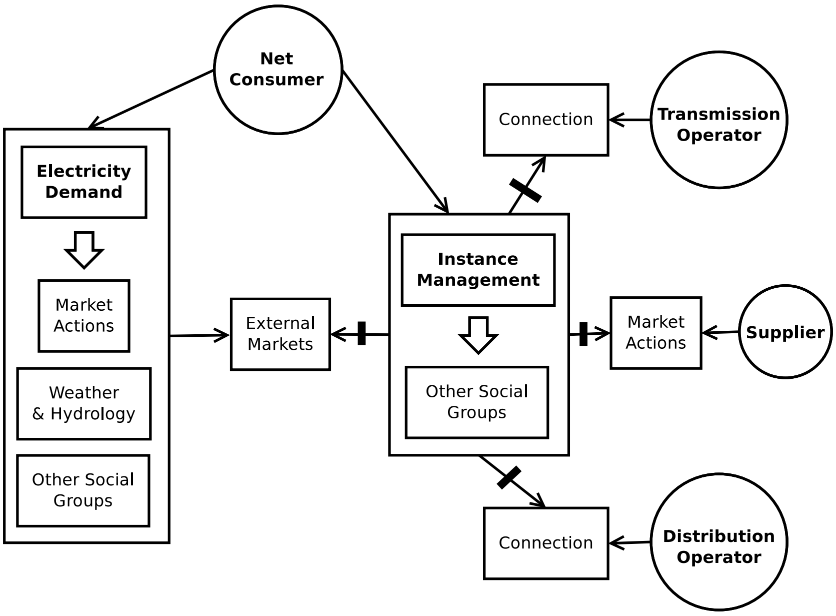

Fig. 5. The fit of (a component of) an example model to the Net Consumer sub-models shown in figure 4. The symbols used are as in figure 3.

figures, which thus relate only to household changes and do not separate out household level decisions from the social groups those households are part of. Therefore, we can take the model as making an assumption of **no-influence** (a 'null assumption') for the industrial-user-related dependencies, and we have to merge Instance Management with Other Social Groups.

In terms of the Electricity Demand sub-model, the relative gas price term relates to the External Markets dependency (the gas market). This models the fuel price drivers of shifts from electricity to gas (or vice versa) in terms of both domestic and industrial users. The remainder (fitted constant) is not separable into any other sub-models, it being a black-box numeric fit. This therefore merges 'unknowable' assumptions from all other dependencies, but the Electricity Demand sub-model still remains a well-defined thing in itself.[16]

Note how Other Social Groups is split; i.e., different aspects of behaviour relating to it are included in separate merged sub-models.

The modelling process more formally defines the different types of fit which are possible. As a very brief further example, imagine if this demand function was just a linear function over time (with the angle and intercept fitted to historical data). A single role instance still exists, but the actions of consumers are not separable from the external changes which affect them, nor from changes in the set of consumers. Thus, there is a single new 'artificial' sub-model which merges both Net Consumer sub-models and all their dependencies.

4 Conclusions

We have outlined the rationale for the approach and the main design principles (via some examples). Outside of this paper, we have currently (a) defined the process, and the required implementation features, in detail; (b) tested it in conceptual fits to several published models; and (c) developed partial implementations of the software framework and some domain-model-conformant models. This has currently focused on the electricity markets domain, but we are working on applying it to a health and social care domain.

Future Work. There are still some features to build into the implementation, which we hope to release as open source. Given the approach's inherent abstraction and generality, there is also a clear need for future work to publish detailed, practical examples which give demonstrable benefits in various areas: conceptual comparison, especially when transdisciplinary; model reuse and alignment; and engagement with policy-makers. With regards to the latter, our ultimate aim is to help move towards model-centric, comparative debate (*à la* Lempert et al. [12]), using scenario generators encapsulating theoretical plurality. The fine-level detail of the process is also difficult to capture in a single paper, so we intend on

[16] There is always some subjectivity in this fitting process. We assume here, for example, that there are no other external-market-related factors which could be folded into this fitted constant.

releasing a 'user-guide-style' technical report which provides a definitive working reference.

Our object-oriented domain model design does not seem particularly commensurate with paradigms such as system dynamics, but we currently believe that it is usable. At the conceptual level, this boils down to translating the abstract system-level attributes (stocks) into the entities that they relate to, with their flows relating to workflows. This will typically involve many merges, which just reflects the general abstraction of system dynamics from individual real-world entities. The implementation level is trickier, since the model is a set of continuous coupled differential equations (albeit resolved numerically with an implicit dependency graph), but we have some ideas on things we can do here which we hope to try out soon.

Difficulties with the 'Atheoretical' Design Criteria. There are potential communication difficulties due to the slippery meaning of 'social theory'. Our restrictions on what can go into a domain model (and how models are mapped to it) obviously constitute a theoretical framework in one sense, and the things in a *particular* domain model constitute some view on the (partial) structure of a real-world system which one could label a social theory (despite our informally-defined intention to include 'as little social theory as possible').

The bottom line is that the domain model tries to include elements that it would be hard to argue as not existing and generalisable in the real-world, whether one wishes to label this as a theory or not. Attempting to fit existing models will show how much their conception differs from this partial base one. If *all* models fitted end up with very large numbers of merges and new entities, this *might* suggest that our domain model is lacking in some regard. Or it might suggest that existing models are all based on very abstract theory as regards the system's structure (which may or may not be a bad thing). The process at least forces us to make these judgements and compare with some more 'common sense' view, which is a valuable scientific endeavour in itself (and promulgates things such as shared vocabularies in the process).

References

1. van der Aalst, W., van Hee, K.: Workflow management models, methods, and systems. MIT Press (2002), `http://www.worldcat.org/isbn/9780262720465`
2. Axelrod, R.: Advancing the art of simulation in the social sciences. In: Conte, R., Hegselmann, R., Terna, P. (eds.) Simulating Social Phenomena, Lecture Notes in Economics and Mathematical Systems, vol. 456, pp. 21–40. Springer (1997)
3. Axtell, R., Axelrod, R., Epstein, J., Cohen, M.: Aligning simulation models: A case study and results. Computational & Mathematical Organization Theory 1(2), 123–141 (1996)
4. Bankes, S.: Improving the utility and the rigor of agent-based modeling through ensembles of models. In: Proceedings of Agent 2003: Challenges in Social Simulation. pp. 155–168 (Oct 2003)
5. Chappin, E.J.L., Dijkema, G.P.J.: Agent-based modelling of energy infrastructure transitions. International Journal of Critical Infrastructures 6(2), 106–130 (2010)

6. DeRemer, F., Kron, H.: Programming-in-the large versus programming-in-the-small. In: Proceedings of the international conference on Reliable software. pp. 114–121 (1975)
7. Edmonds, B., Hales, D.: Replication, replication and replication: Some hard lessons from model alignment. Journal of Artificial Societies & Social Simulation (JASSS) 6(4), 11 (2003), http://jasss.soc.surrey.ac.uk/6/4/11.html
8. Evans, E.: Domain-Driven Design: tackling complexity in the heart of software. Addison-Wesley (2004)
9. Fowler, M., Rice, D., Foemmel, M., Hieatt, E., Mee, R., Stafford, R.: Patterns of Enterprise Application Architecture. Addison-Wesley (2003)
10. ter Hofstede, A., van der Aalst, W., Adams, M., Russell, N.: Modern Business Process Automation: YAWL and its Support Environment. Springer (2010)
11. Lempert, R.: A new decision sciences for complex systems. Proceedings of the National Academy of Sciences (PNAS) 99, 7309–7313 (2002)
12. Lempert, R., Popper, S., Bankes, S.: Shaping the next one hundred years: new methods for quantitative, long-term policy analysis. Tech. Rep. MR-1626, RAND Corporation (2003)
13. Luke, S., Cioffi-Revilla, C., Panait, L., Sullivan, K., Balan, G.: MASON: A multi-agent simulation environment. Simulation 81(7), 517–527 (2005)
14. Moss, S.: Relevance, realism and rigour: A third way for social and economic research. Tech. Rep. 99, Centre for Policy Modelling (CfPM), Manchester Metropolitan University (1999)
15. Moss, S.: Alternative approaches to the empirical validation of agent-based models. Journal of Artificial Societies & Social Simulation 11(1), 5 (2008), http://jasss.soc.surrey.ac.uk/11/1/5.html
16. Parker, D.C., Brown, D.G., Polhill, J.G., Deadman, P.J., Manson, S.M.: Illustrating a new "conceptual design pattern" for agent-based models of land use via five case studies—the MR POTATOHEAD framework., chap. 2, pp. 23–51. INSISOC, Spain (2008)
17. Polhill, J.G., Parker, D., Brown, D., Grimm, V.: Using the ODD protocol for describing three agent-based social simulation models of land-use change. Journal of Artificial Societies & Social Simulation 11(2), 3 (2008), http://jasss.soc.surrey.ac.uk/11/2/3.html
18. Richiardi, M., Leombruni, R., Saam, N., Sonnessa, M.: A common protocol for agent-based social simulation. Journal of Artificial Societies & Social Simulation 9(1), 15 (2006), http://jasss.soc.surrey.ac.uk/9/1/15.html
19. Rossiter, S., Bell, K.R.W.: A workflow hybrid as a multi-model, multi-paradigm simulation framework. In: Janssens, G.K., Ramaekers, K., Caris, A. (eds.) ESM 2010: The 2010 European Simulation and Modelling Conference. pp. 37–41. Eurosis (2010)
20. Rouchier, J., Cioffi-Revilla, C., Polhill, J., Takadama, K.: Progress in model-to-model analysis. Journal of Artificial Societies & Social Simulation 11(2), 8 (2008), http://jasss.soc.surrey.ac.uk/11/2/8.html
21. Sterman, J.: Business Dynamics: Systems Thinking and Modeling for a Complex World. McGraw-Hill (2000)
22. Sun, J., Tesfatsion, L.: Dynamic testing of wholesale power market designs: An open-source agent-based framework. Computational Economics 30(3), 291–327 (Oct 2007)
23. Zeigler, B.P., Gon Kim, T., Praehofer, H.: Theory of modeling and simulation : integrating discrete event and continuous complex dynamic systems. Academic Press, 2nd edn. (2000)