Querying Histories of Organisation Simulations

Tony Clark *t.clark@shu.ac.uk*
Sheffield Hallam University
Sheffield, UK

Balbir Barn *b.barn@mdx.ac.uk*
Middlesex University London
London, UK

Vinay Kulkarni *vinay.vkulkarni@tcs.com*
Tata Consultancy Services Research
Pune, India

Souvik Barat *souvik.barat@tcs.com*
Tata Consultancy Services Research
Pune, India

Abstract

Industrial Dynamics involves system modelling, simulation and evaluation leading to policy making. Traditional approaches to industrial dynamics use expert knowledge to build top-down models that have been criticised as not taking into account the adaptability and sociotechnical features of modern organisations. Furthermore, such models require a-priori knowledge of policy-making theorems. This paper advances recent research on bottom-up agent-based organisational modelling for Industrial Dynamics by presenting a framework where simulations produce histories that can be used to establish a range of policy-based theorems. The framework is presented and evaluated using a case study that has been implemented using a toolset called ESL.
Keywords: Simulation, Theory building mechanism

1. Introduction

In his work on *Industrial Dynamics*, Forrester [9] identifies four fundamental steps: (1) modelling a system; (2) simulation; (3) evaluation; (4) policy making. When applying these principles to organisations, it is typical to use expert knowledge about key aspects of the business to build a model expressed as equations derived using mathematical techniques such as stochastic processes [17], genetic algorithms [5] and system dynamic models [19]. Given such a model, simulation is performed by setting initial values and then running the equations for a particular time period. The final state of the model can be evaluated against real-world data and the expectations of domain experts. Finally, when the model has been validated it can be used to predict future system states that influence policy and strategy decisions.

This approach is based on the assumption that the model of an organisation can be expressed using equations representing behaviour that is stable and linear in nature. It has been argued [15] that such a behaviour is much more dynamic, adaptive, complex and non-linear. Where the nature of modern organisations is represented as a collection of distributed autonomous socio-technical entities with emergent behaviour that does not lend itself to traditional top-down methods. Further bottom-up behavioural modelling [18], such as that used in multi-agent systems [7], is more appropriate for the analysis of modern organisations [20].

There is increasing interest in using data analytics for decision-making [10]; augmenting real-world with synthetic data before looking for patterns. However, such top-down automated data-driven analyses are risky[1] and can only establish co-relations, not causality, and even then are vulnerable to sampling errors. Moreover, such data-driven techniques can at best do linear extrapolation of what has happened so far, and hence are fundamentally not equipped to handle open-ended problem spaces [6, 10]. Therefore, complex decision-making tasks depend on both simulation and human expertise, and we seek approaches that support decision-making by combining simulation, data analysis and domain-specific expertise.

Our hypothesis is that it is practical and effective to take an *emergent behaviour* approach to the construction of simulation models whereby the appropriate elements of the organisation are represented as autonomous agents. Judicious construction of such models will allow a human decision-maker to observe simulation runs, make interactive modifications and to infer the relationships that lead to goal maximisation. This raises several methodological questions in the context of decision-making: (1) how to construct appropriate actor models; (2) how to construct a suitable execution engine; (3) how to evaluate the results of actor-based simulations. Our previous work in this area has proposed an actor-based language called ESL for (1) and (2); this paper proposes a novel approach to (3) in terms of constructing *queries* over *simulation histories*. Our contribution is to propose a framework that integrates simulations, histories, and queries and to demonstrate that this can be implemented using ESL.

2. Related Work

Understanding the behaviour of an organisation largely depends on the knowledge of domain experts within the context of a dynamic business environment. Such knowledge is based on theories that are constructed and tested through experimental design and empirical data that exists within an organisation. These theories have specific epistemological construction and purpose that centres around four key components: constructs that are the basic measurable conceptual elements extracted from the domain of discourse; relations that describe connections among constructs and their interactions with one another; boundaries describing the scope or the validity of the theory under certain conditions; and propositions being statements that are concerned with making predictions about a theory's constructs [21, 23, 26]. The purpose of such a theory includes *analysis*, *explanation*, *prediction*, *explanation-and-prediction*, and *design and action* [11].

Organisational theories discussed in this paper are limited to the analysis and prediction of socio-technical systems that are largely uncertain and non-linear in nature. Moreover, sufficient empirical data to test organisational and behavioural theories are challenging to obtain in this context and therefore we argue a grand theory is infeasible. This is a problematic situation since traditional experimental research that is based on empirical data is not suitable due to data limitations [28] and is also reported as being an ineffective approach for strategy and policy-making since the relevance of historical data in the context of a dynamic environment is questionable [6, 10]. On the other hand, the experimental research that uses simulation introduces complexity around the epistemic value of a theory [12].

The use of simulation in theory building and validation is criticised as relying on a partial representation of the real world, and thus a theory derived from a simulation is essentially a theory about the model underlying the simulation. Moreover, the analysis and prediction of socio-technical systems is a case where theories occupy a position that requires both positivist and interpretivist paradigms due to the inherent characteristics of the system and system histories. Taking a reductionist position, we argue that the empirical data limitation of socio-technical systems from a positivist position is an unsolvable problem whereas the concerns of a simulation based approach can be fixed by constructing model with appropriate *constructs*, *relations*, *boundaries* and *propositions*.

[1] http://sloanreview.mit.edu/article/why-big-data-isnt-enough/

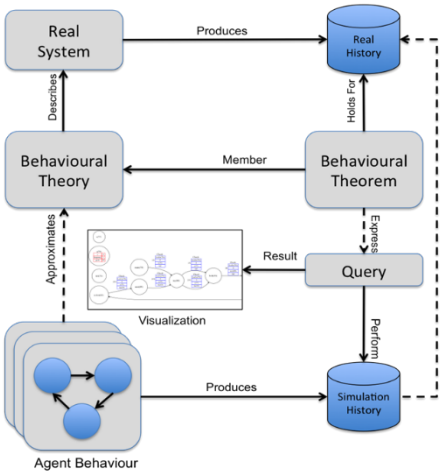


Fig. 1: Architecture

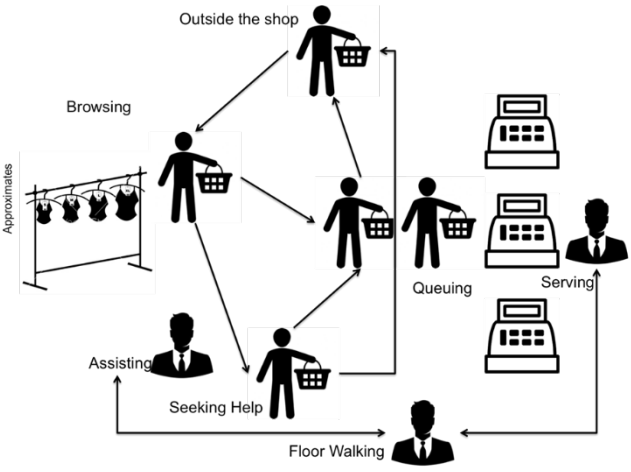


Fig. 2: Shopping Simulation

Though it is a controversial topic, a simulation based approach for theory building and validation is discussed extensively in strategy and policy making in the context of complex organisations. For instance, March [17] and Davis *et al* [8] recommend stochastic process based simulation to validate theoretical logic, Lomi and Lersen [16] propose cellular automata to understand emergence of spatial relationships, Zott [27] and Bruderer and Singh [5] propose genetic algorithm based simulation for constructing and validating *organisational theory*. Schwaninger et al. [19] recommend system dynamic model centric simulation for developing and validating *organisational theory* and *behavioural theory*. The experimental setup for stochastic processes, genetic algorithms and system dynamic models uses a top-down approach for constructing enterprise models; essentially they use theory building activities to construct system models and consider known facts as the *proposition* in a simulation setting to deduct or validate unproven (grand) theories.

Our approach models the known micro-behaviour of a system as a collection of agents and observes emergent macro-behaviour through multiple simulation runs, captured using histories, to construct and validate theories. Earlier work describes using actor-based technologies [1] to encode agent simulations for decision-making [4, 14] but lacks support for the representation and validation of theories. Existing actor-based technologies [3, 25, 24, 22, 13, 2] also lack support for theories which has led to the development of a new actor-based language called ESL whose support for theory-building is described in this paper.

The high-level architecture of the proposed approach involving agent-based simulations, histories, theories and queries is described in section 3. A simple case study is described in section 4 and is used to motivate a domain analysis of the key features of simulation that leads to a representation for simulation histories in section 5. A technology for expressing queries over histories is described in section 6 that are demonstrated to be effective with respect to the case in section 7. Section 8 describes our approach to implementing histories and queries in ESL, some of the limitations of the work, and outlines future plans.

3. Proposed Architecture and Domain Analysis

Figure 1 shows an overview of the proposed architecture that uses bottom-up behavioural modelling to simulate organisations. Consider a *real system* running in an ideal situation where all possible states are recorded in a history. An expert understanding of such a system is shown as a *behavioural theory*. In general it is not possible to produce a formal representation of the behavioural theory because, as we have noted above, this is complex, non-linear and exhibits socio-technical features. However, experts can propose individual *behavioural theorems* that could be verified against the history.

Based on our proposal, an organisation is analysed in order to identify the component agents and their associated behaviour leading to a simulated system that produces a

simulation history. Each behavioural theorem can then be checked against the history in order to validate it. A sufficient collection of satisfied theorems provides confidence that the agent-based system is consistent with the real system.

The proposal leads to an interative approach that is outlined as follows: (1) identify the actors in the system; (2) model their behaviours; (3) run and capture the history; (4) formulate a theory about the system; (5) list particular theorems that should hold; (6) express each theorem as a query; (7) test that the theorem holds by running the query against the history. This approach requires a suitable actor-based behavioural modelling language, a model of behavioural histories and an associated query language. We have developed a modelling language called ESL that is used to evaluate the approach. All the examples given in this paper have been implemented in ESL and are available via the ESL repository[2].

4. Case Study

The approach described in section 3 will be evaluated against a simple case study that is based on existing work on agent-based organisation simulations [20] and is shown in Figure 2. A shop provides stock on the shop-floor. Customers enter the shop and may browse until they either leave, seek help or decide on a purchase. Items must be purchased at tills and multiple customers are serviced via a queue. Shop assistants may be on the shop-floor, helping a customer or may service a till. A queueing customer can only make a purchase when they reach the head of a queue at a serviced till. A customer who waits too long at an un-serviced till, or for whom help is not available, will become unhappy and leave the shop. The shop would like to minimise unhappiness.

The key actors in the system are shown in figure 2 and their behaviours can be defined as an elaboration of the associated flows. Section 5 uses the case study as the motivation for a domain analysis of actor-based behaviour leading to the design of actor histories and their associated theories and queries. The system behaviour emerges from the combination of the individual behaviours and the actor interactions, describing, for example, how a customer who is seeking help becomes advised by a shop assistant and then joins a till-queue to purchase an item. Particular theorems can then be formulated in terms of the proposed theory. A simple theorem proposes that customers who wait for longer than a given time at a till-queue will leave the shop without making a purchase. Such a theorem is of interest when trying to establish business goals involving customer satisfaction. A more complex theorem involves the detection of potentially criminal behaviour whereby all assistants have been engaged by customers seeking help at the same time as a customer leaves the shop. We might reasonably consider this to be suspicious, especially if we have reason to believe that a criminal gang is operating in the area and if we have noticed stock to be going missing.

5. Behavioural Representation

Figure 1 proposes that simulation histories are a representation of the emergent behaviour produced by a collection of interacting agents. The behavioural representation encodes expert theories about behaviour in the organisational domain of interest. Such expert theories must be encoded using a *meta-theory* (in this case agents) which is described as follows: section 5.1 motivates the key features of agent behaviour; section 5.2 encodes the case study as a collection of actors; section 5.3 describes the structure of the histories that result from agent simulations.

5.1. Domain Analysis

[2] https://github.com/TonyClark/ESL

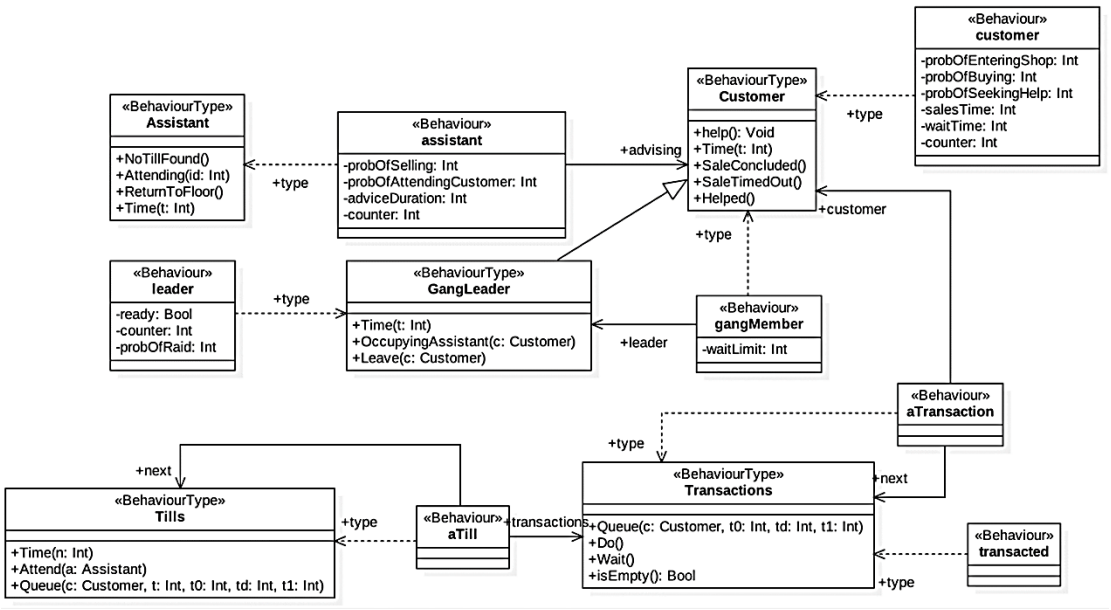


Fig. 3: Case Study Agents

We propose that any organisation can viewed as a collection of interacting autonomous agents. The agent concept must support both atomic and aggregate behaviour and autonomy implies that agents may only influence each other through communications, *i.e.,* it is not possible for one agent to completely dominate the behaviour of another. Methodologically, we seek to express an organisation of interest in terms of a collection of atomic agents, including IT systems, organisational units and people, whose individual and collective activities and communications gives rise to emergent system behaviour.

Individual agent behaviour is driven by local goals. An agent goal is a condition on its state history that influences its responses to external stimuli including the passage of time. Typically an agent will want to improve the key performance indicators (KPIs) of the organisation, including its local state and its environment, in order to achieve its goal that may change over time.

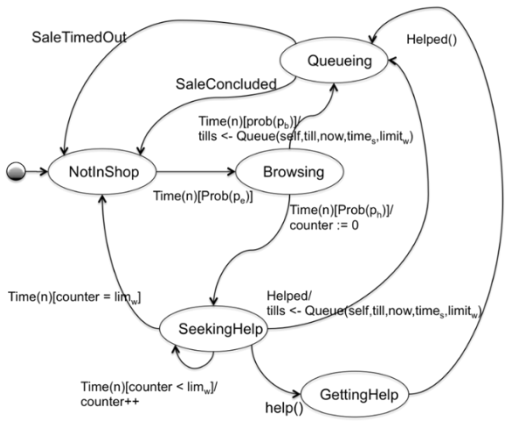
A simulation that supports decision-making will often involve an overall goal (for example maximise profit or minimise cost). Such a goal is similar to an agent's goal in that it can be expressed in terms of KPIs on system histories. Methodologically, it may be tempting to decompose a system goal into agent goals, however this is not necessarily the case, since we must acknowledge that an organisation is a socio-technical system where social features give rise to self-interest.

A simulation must take into account stochastic behaviour whereby individual agent activity has variation controlled by probability. Expert theories about the organisational domain together with real-world measurement will provide the basis for encoding the details such as particular probability distributions.

A simulation will include those features that can vary, its *levers*, and those things that are fixed. Levers differ from probabilities, in that a lever is something that is fixed at the start of simulation, but may differ from run to run.

Finally, a simulation must have some mechanism for synchronising behaviour in terms of time. Typical examples of simulation requirements include jobs being completed on time, or minimising the amount of time spent on certain tasks. We will assume that there is a global clock that broadcasts the current time to all actors.

The domain analysis outlined above represents the requirements for an actor-based simulation language called ESL that consists of behaviour, histories and queries. This paper will not give details of ESL behaviour because our focus is on histories and queries; behavioural representation will be given in terms of actor models expressed in terms of type structures and state-machines.



(a) Customer Machine

```
act name(x:Type)::BehaviourType {
  export a;          // public interface.
  a::Int = 0;        // public variable.
  b::A = new A(x);   // initialised local.
  s::State = S0;     // state variable.
  Time(t::Int)       // message handler.
    when s = S0 and
         a < 10      // message guard.
  → {                // message action.
    a := a + 1;      // update variable.
    s := S1;         // change state.
    b ← M(t)         // send message.
  }
}
```

(b) Example of ESL behaviour

Fig. 4: Customer Behaviour

5.2. Actor Behaviour

Organisation behaviour consists of a collection of autonomous actors that communicate using asynchronous messages. A history will record the behavioural concepts and relations during execution. This section provides a concrete description of an actor-based behaviour in order to motivate the design of a history model in the following section.

The behaviour of an actor can be described using a state-machine and an associated actor interface. The interface of an actor consists of state variables and message handlers. A state variable may be a simple value or may be a reference to another actor. The structure of the case study is shown in figure 3 and a single state-machine is shown in figure 4. The rest of this section describes the behaviour of a customer in terms of these models and outlines how the behaviour is encoded in ESL.

The initial state of a customer is NotInShop. The simulation regularly sends Time(t) messages to all actors. On receiving the current time, a customer who is not in shop may, with a given probability, enter the shop at which point their state changes to Browsing. Further time messages will change the state of a customer to either SeekingHelp or Queueing depending on associated probabilities. When an assistant receives a time message it will check for customers seeking help; if one is found (by synchronously interrogating a customer's state), then the state of the customer changes to GettingHelp via a synchronous help message sent from the assistant to the customer. Once help is concluded, the assistant will send the customer a Helped message that changes the customer's state to Queueing. On changing to a Queueing state the customer will inform the appropriate till via a Queue message and will start to wait. The till will receive Time messages and will process the customer at the head of the queue by creating a transaction that will last for a proscribed period of time before changing to transacted and informing the customer via a SaleConcluded message. If a customer waits at a time for a period of time that exceeds waitTime then the customer received a SaleTimedOut message and its state changes to NotInShop. Similarly, waitTime governs when a customer who is waiting for help will leave the shop.

A criminal gang consists of members and a leader, all of whom have customer behaviours. A gang leader extends Customer with a message from each gang member OccupyingAssistant(c) that is used to track the progress of the gang that is trying to occupy all assistants: once this is achieved the leader can strike.

ESL provides a statically typed language for encoding actor-based simulation models. Since our focus is on histories and queries, this paper does not provide the details of behavioural encoding of figures 3 and 4, however figure 4b shows an example ESL actor definition including a message handler that represents a state-machine transition[3].

[3] implementation at github.com/TonyClark/ESL/blob/master/esl/shop.esl

Name	Type	Example	Description
actor	(Id,Str,Time)	actor(1,'customer',2)	A new actor is created.
state	(Id,Str,Value,Time)	state(1,'counter',1,10)	Actor state is changed.
send	(Id,Id,Value,Time)	send(0,1,Time(2),2)	Message is sent.
consume	(Id,Value,Time)	consume(1,Time(2),2)	Message is handled.
become	(Id,Str,Str,Time)	become(2,'aTransaction','transacted',3)	Behaviour change.

Fig. 5: History Facts

5.3. Histories

Expert domain knowledge can be used in two different ways: to encode a behavioural theory about an organisation in terms of an actor-based meta-theory, and to encode queries arising from the behavioural theory relating to decision-making. The interface between these two use-cases is a behavioural history that is generated by the simulation and subsequently processed by the query. The behavioural model described in the previous section produces a history as a set of facts defined in Figure 5. Actor identifiers of type Id are integers, the type Time is a synonym for integer, and the type Value is a union of the atomic types Int and Str, lists of values, and terms of the form T(v,…).

6. Theorems and Queries

The previous section has described how ESL generates execution histories. Expert theories and queries about the organisational behaviour may concretely take a number of different forms depending on the intended use. ESL provides a query language based on Prolog that has been extended with temporal features that can range over the information in the histories. A theory about a simulation is encoded as a collection of Prolog-style rules and a theorem is encoded as query.

An ESL rule takes the form of fact←query,… where fact is established providing that the sequence of queries are satisfied. The following example shows how lists are processed to remove all elements:

```
removeAll ([] ,vs ,vs);
removeAll ([x|xs],vs ,vs '')← remove (x,vs ,vs '),removeAll (xs ,vs ',vs '');
remove (x ,[] ,[]) ;
remove (x ,[x | xs], result )← remove (x,xs , result );
remove (x ,[y|xs ],[y| result ])← remove (x,xs , result );
```

Figure 6 shows the different types of ESL query. Each query is said to be *satisfied* with respect to a history at a *given time t*. A fact n(e,…) is satisfied if there is a rule whose head matches n(e,…) and whose body is satisfied. Facts involving actor, state, become, send and consume directly interrogate the history at the current time with respect to events that occurred. Queries next and prev involve sub-queries that must hold at the next and previous

Query	Satisfaction Criteria
n(e...)	The fact n(e,...) holds in the history at the current time.
actor(i,b,t)	An actor was created with unique identifier i, behaviour b at time t.
state(i,n,v,t)	Actor i has state n=v at time t.
become(i,b,t)	Actor i changes behaviour to b at time t.
send(i1,i2,m,t)	Actor i1 sends message m to i2 at time t.
consume(i,m,t)	Actor i processes message m at time t.
next[q]	q is satisfied at the next time unit in the history. Fails if currently end of time.
prev[q]	q is satisfied at the previous time in the history. Fails if currently start of time.
always[q]	q holds in the history at the current time and at all future times.
eventually[q]	q holds in the history at the current time or at any future time.
past[q]	q holds in the history at the current time or at any past time.
start	Tthe current time is the beginning of the history.
end	The current time is the end of the history.
forall[q](e,l)	satisfy q all possible ways that produce e in list l.

Fig. 6: The ESL Query Language

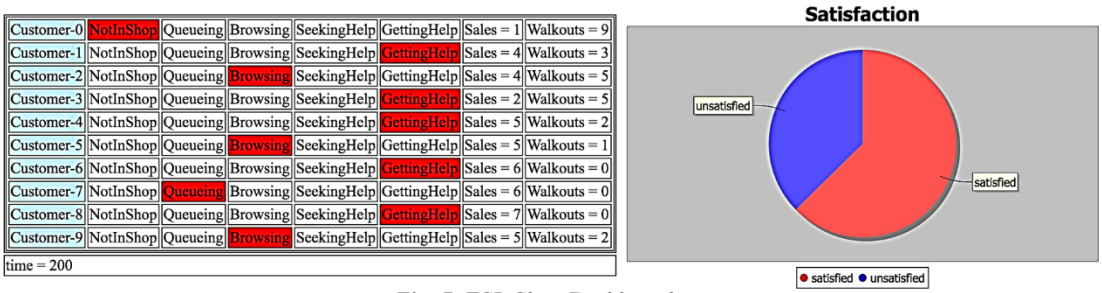


Fig. 7: ESL Shop Dashboard

time instant respectively. For example next[actor(i,'customer',t)] is satisfied when a customer is created at the next time instant.

Queries involving always must hold for all future times, for example always [state(i,x,10,_)] is satisfied when the actor with identity i has a state variable x whose value remains 10 for the rest of the history; whereas eventually[state(i,x,10,t)] is satisfied when the actor with identity i has a state variable x that becomes 10 at some time t in the future.

Queries involving forall are used to gather together all the possible ways in which a fact can be satisfied. For example forall[eventually[actor(i,'customer',_)](i,customers) is satisfied when customers is a list of all the customer actors.

7. Demonstration of the Approach

Our proposition is that simulations can be expressed using configurations of actors that implement agent systems and that theorems about the simulation can be expressed as queries over simulation histories. To evaluate the proposition we have extended the actor-based language ESL with histories and queries and implemented the case study described in section 4. The case study has been simulated with a variety of configurations up to 50 customers, 10 tills and 7 assistants over 1000 steps with a corresponding 9MB history. Figure 7 shows the final view of the real-time dashboard produced by the ESL simulation based on 10 customers, 4 tills, 5 assistants and 200 steps.

Domain theories are translated into ESL rule-sets over histories as described in section 6 and domain theorems are translated into ESL queries. This section provides examples of three queries: section 7.1 defines a history property that represents all the customers who fail to make a purchase; section 7.2 defines a property that the shop manager believes to represent a raid by a criminal gang; section 7.3 defines a mapping from a history to a graphical representation that allows an expert to detect interesting features of a simulation. The ESL implementation of the queries is available as part of the ESL repository[4].

7.1. No Purchases

A customer fails to make a purchase when there is no record of the actor processing a SaleConcluded message. The following rule-set defines a property of a history noSales(cs) where cs is a list of customer identifiers that have not processed a SaleConcluded message:

```
customers (cs) ← forall [eventually[actor(a,'customer ',_ ) ]](a,cs);
makesPurchase ([] ,[]) ;
makesPurchase ([c | cs ],[c | cs ']) ← makePurchase (c), makesPurchase (cs ,cs ');
makesPurchase ([_ | cs],cs ') ← makesPurchase (cs ,cs ');
makePurchase (c) ← eventually[send( _,c, SaleConcluded ,_ )];
noSales (cs) ← customers (c), makesPurchase (c,p), removeAll [ Int ](p,c,cs);
```

[4] https://github.com/TonyClark/ESL/blob/master/esl/query/shop.q

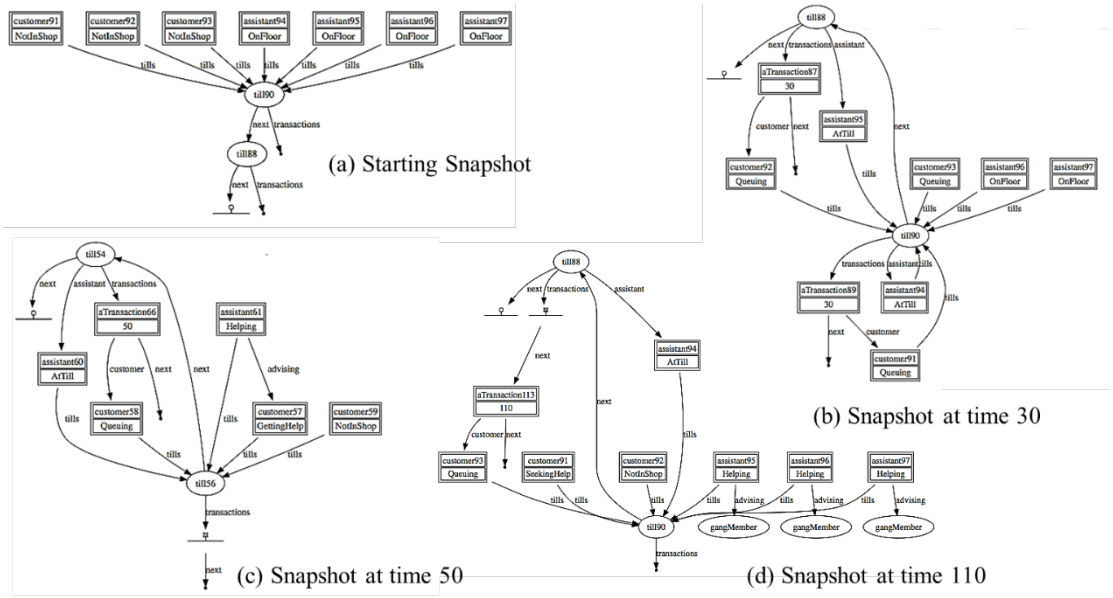


Fig. 8: A History Filmstrip

The customers rule calculates the identifiers of all customers that are created in the history. The eventually fact is repeatedly satisfied by forall, gathering up the identifiers a and return them as a list cs. The makePurchase rule is satisfied for a customer c providing the history contains a sale-concluding message. Finally, noSales holds for a list of customers cs providing that they do not receive the SaleConcluded message.

7.2. Detecting a Raid

The shop manager suspects that the shop is losing merchandise to a gang operating in the area. The manager's theory is that the gang operates by occupying all the assistants while the gang leader leaves the shop with stolen goods. The following rule-base expresses this theory over a history:

```
1 raid ( assistants , Raid ( time ))
2   forall [eventually[actor(a,'assistant ',_)]](a, assistants ),
3   eventually[ allHelping ( assistants , time )], !;
4 raid ([] , NoRaid );
5
6 allHelping ([] ,_);
7 allHelping ([a|as],t)   state(a,'advising ',Ref (_),t), allHelping (as ,t);
```

The fact allHelping(as,t) (lines 6-7) is true when each of the assistants as is advising a customer at time t. The term Ref(i) represents a reference to an actor. A raid occurs at time t, raid(assistants,Raid(t)) (lines 1-4) when all the assistants busy helping customers. Notice that line 3 eventually establishes the fact allHelping(assistants,time). If no raid can be established then the history entails the fact raid([],NoRaid).

7.3. History Filmstrips

Expert theories about a domain may need to be developed by interacting with a simulation. This can be difficult to include in a simulation definition because, by its nature, theory development is opportunistic. Simulation histories can facilitate interactive development of a theory by presenting filmstrips that consist of individual simulation snapshots.

ESL supports the construction of filmstrips by mapping a history to a sequence of display elements that represent time-order. By displaying the elements in sequence, the history can be

```
1  filmstrip(filmstrip) ← customers(cs), filmstrip(cs,filmstrip);
2
3  filmstrip(_,[]) ← end;
4  filmstrip(cs,[s | f]) ← snapshot(cs,s), next[filmstrip(cs,f)];
5
6  snapshot(cs,Pie([Slice([],'satisfied',sat),Slice([],'unsatisfied',unsat)])) ←
7    getSales(cs,sales), length(cs,l), length(sales,sat), unsat := l - sat;
8
9  getSales([],[]);
10 getSales([c|cs],[c|ss]) ← past[send(_,c,'SaleConcluded',_)], getSales(cs,ss);
11 getSales([_ | cs],ss) ← getSales(cs,ss);
```

Fig. 9: Filmstrip Construction

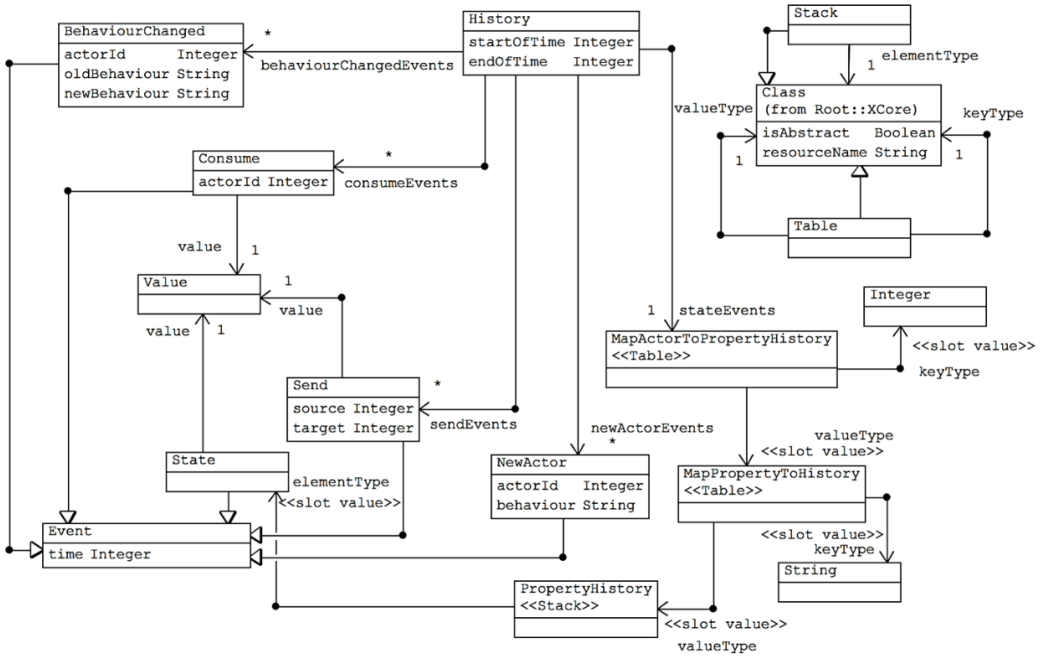


Fig. 10: ESL History Implementation

played forward and backwards. Various display element types are available including Graph that consists of nodes and edges, Table, and Pie that represents a pie-chart.

Figure 8 shows 4 snapshots generated by ESL at different times during a simulation of the shop case study[5]. The graph vertices and edges are a projection of the model shown in figure 3. Figure 8a shows the start of the simulation containing 3 customer, 4 sales assistants and 2 tills (the criminal gang is not shown but consists of 3 members and a leader). Figure 8b shows the situation at time 30 where two customers are being served at different tills and the third customer is waiting. Different assistants are serving both tills and the other two assistants are both on the floor. Figure 8c is a snapshot at time 50 where one customer has left the shop, another is getting help from an assistant and the third customer is making a purchase. Finally, figure 8d shows three gang members occupying three shop assistants in preparation for a raid.

Figure 9 shows an example theory that produces a filmstrip of customer satisfaction pie-charts. Notice how the filmstrip rule (lines 3-4) steps through the history using next until it reaches the end. Each snapshot is constructed using past to check whether a particular customer has made a sale to date.

8. Analysis and Conclusion

Assuming that organisational simulation in terms of actor-based agent models is a suitable approach, our proposition is that expert analysis of simulation data can be effectively supported through the use of histories and associated queries. Given that no existing actor

[5] The larges images of the filmstrips are available at https://www.dropbox.com/s/m3jbvv2n3tn5lgb/History%20Filmstrip.pdf?dl=0

technology supports this approach, the purpose of this paper is to demonstrate the technical feasibility of the proposition. This has been achieved by analysing the requirements for data representation in a history and by showing how a temporal extension to Prolog can be used to encode queries as an extension to the ESL simulation language.

The implementation consists of modifying the ESL virtual machine to capture history data as defined in figure 5. An important technical issue to be addressed when representing histories generated by realistic simulations is the size of the data and the speed of access to facts. Figure 10 shows the data model for the Java classes that implement ESL histories where indexing has been used to reduce the amount of data stored and speed up reference to facts by queries.

ESL has also been extended with a rule-language as described in this paper. The language is currently interpreted using Prolog unification and uses continuations for backtracking, but will need to be compiled if it is to scale effectively. It is reasonable to expect that standard techniques for compiling Prolog can be used to achieve this. Both the actor behaviour and query languages are statically typed and checked.

The query language described in this paper appears to be suitable for encoding expert theories about a simulation that are based on symbolic processing. Other techniques are likely to be important when interpreting large-scale histories including data mining and machine learning.

This paper defines very general-purpose low-level facts about a simulation history. Our expectation is that the approach will be improved by allowing facts, theories and associated queries to be domain-specific. Furthermore, since ESL is statically typed, it would be possible to perform analysis on both the simulation model and the queries to optimise the information stored in a history by ignoring those facts that will not be required.

The general method for constructing simulations, theories and queries described in section 3 has not been validated against a real-world case study. Although the case study used to drive our technical evaluation in this paper is valid (see [20] for more detail) it was not necessary to employ the method or to expand it, since the details of the application are known a-priori. This is an area for future work that is planned using several real-world case studies provided by TCS.

References

1. Gul A Agha. Actors: A model of concurrent computation in distributed systems. Technical report, DTIC Document, 1985.
2. Jamie Allen. Effective akka. " O'Reilly Media, Inc.", 2013.
3. Joe Armstrong. Erlang - a survey of the language and its industrial applications. In Proc. INAP, volume 96, 1996.
4. Balbir S Barn, Tony Clark, and Vinay Kulkarni. Can organisational theory and multi-agent systems influence next generation enterprise modelling? In International Conference on Software Technologies, pages 202-216. Springer, 2014.
5. Erhard Bruderer and Jitendra V Singh. Organizational evolution, learning, and selection: A genetic-algorithm-based model. Academy of management journal, 39(5):1322-1349, 1996.
6. Sen Chai and Willy Shih. Why big data isn't enough. MIT Sloan Management Review, November 2016.
7. Valentino Crespi, Aram Galstyan, and Kristina Lerman. Top-down vs bottom-up methodologies in multi-agent system design. Autonomous Robots, 24(3):303-313, 2008.
8. Jason P Davis, Kathleen M Eisenhardt, and Christopher B Bingham. Complexity theory, market dynamism, and the strategy of simple rules. In Proceedings of DRUID Summer Conference, pages 18-20. Citeseer, 2007.
9. Jay Wright Forrester. Industrial dynamics. Journal of the Operational Research Society, 48(10):1037-1041, 1997.

10. John Forsyth and Leah Boucher. Why big data is not enough. Research World, 2015(50):26-27, 2015.
11. Shirley Gregor. The nature of theory in information systems. MIS quarterly, pages 611-642, 2006.
12. Till Grune-Yanoff and Paul Weirich. The philosophy and epistemology of simulation: a review. Simulation & Gaming, 41(1):20{50, 2010.
13. Philipp Haller and Martin Odersky. Scala actors: Unifying thread-based and event-based programming. Theoretical Computer Science, 410(2):202-220, 2009.
14. Vinay Kulkarni, Souvik Barat, Tony Clark, and Balbir Barn. Toward overcoming accidental complexity in organisational decision-making. In ACM/IEEE 18th International Conference on Model Driven Engineering Languages and Systems (MODELS), pages 368-377. IEEE, 2015.
15. Jonas Larsson and J Petersson. Agent-based modelling in fashion demand chains. In Proceedings of the 19th Annual NOFOMA Conference, Reykjavik, Iceland, June, volume 7, 2007.
16. Alessandro Lomi and Erik R Larsen. Interacting locally and evolving globally: A computational approach to the dynamics of organizational populations. Academy of Management Journal, 39(5):1287-1321, 1996.
17. James G March. Exploration and exploitation in organizational learning. Organization science, 2(1):71-87, 1991.
18. Markus Pizka and Andreas Bauer. A brief top-down and bottom-up philosophy on software evolution. In Software Evolution, 2004. Proceedings. 7th International Workshop on Principles of, pages 131-136. IEEE, 2004.
19. Markus Schwaninger and Stefan Groesser. System dynamics as model-based theory building. Systems Research and Behavioral Science, 25(4):447-465, 2008.
20. Peer-Olaf Siebers and Uwe Aickelin. A first approach on modelling staff pro-activeness in retail simulation models. J. Artificial Societies and Social Simulation, 14(2), 2011.
21. Dag IK Sjoberg. Documenting theories working group results. In Empirical Software Engineering Issues. Critical Assessment and Future Directions, pages 111-114. Springer, 2007.
22. Sriram Srinivasan and Alan Mycroft. Kilim: Isolation-typed actors for java. In European Conference on Object-Oriented Programming, pages 104-128. Springer, 2008.
23. Klaas-Jan Stol and Brian Fitzgerald. Uncovering theories in software engineering. In Software Engineering (GTSE), 2013 2nd SEMAT Workshop on a General Theory of, pages 5-14. IEEE, 2013.
24. Tom Van Cutsem, Stijn Mostinckx, Elisa Gonzalez Boix, Jessie Dedecker, and Wolfgang De Meuter. Ambienttalk: object-oriented event-driven programming in mobile ad hoc networks. In Chilean Society of Computer Science, 2007. SCCC'07. XXVI International Conference of the, pages 3-12. IEEE, 2007.
25. Carlos Varela and Gul Agha. Programming dynamically reconfigurable open systems with salsa. ACM SIGPLAN Notices, 36(12):20-34, 2001.
26. Roel Wieringa, Maya Daneva, and Nelly Condori-Fernandez. The structure of design theories, and an analysis of their use in software engineering experiments. In Empirical Software Engineering and Measurement (ESEM), 2011 International Symposium on, pages 295-304. IEEE, 2011.
27. Christoph Zott. When adaptation fails an agent-based explanation of inefficient bargaining under private information. Journal of Conflict Resolution, 46(6):727-753, 2002.
28. Christoph Zott. Dynamic capabilities and the emergence of intra-industry differential firm performance: insights from a simulation study. Strategic management journal, 24(2):97-125, 2003.