OrgML - A Domain Specific Language for Organisational Decision-Making

Souvik Barat, Balbir Barn, Tony Clark, Vinay Kulkarni

OrgML - A domain specific language for organisational decision-making

Souvik Barat[1], Balbir Barn[2],
Tony Clark[3], and Vinay Kulkarni[1]

[1] Tata Consultancy Services Research, Pune, India
{souvik.barat,vinay.vkulkarni}@tcs.com
[2] Middlesex University London, UK
b.barn@mdx.ac.uk
[3] Aston University, Birmingham, UK
tony.clark@aston.ac.uk

Abstract. Effective decision-making based on precise understanding of an organisation is critical for modern organisations to stay competitive in a dynamic and uncertain business environment. However, the state-of-the-art technologies that are relevant in this context are not adequate to capture and quantitatively analyse complex organisations. This paper discerns the necessary information for an organisational decision-making from management viewpoint, discusses inadequacy of the existing enterprise modelling and specification techniques, proposes a domain specific language to capture the necessary information in machine processable form, and demonstrates how the collected information can be used for a simulation-based evidence-driven organisational decision-making.

Key words: Organisational decision making, Enterprise Modelling, Enterprise Simulation, Domain Specific Language, What-if Analysis.

1 Introduction

Modern organisations continuously evaluate their status-quo and evolve to stay competitive and *economically viable* in the current business environment [14]. In this endeavour, decision-makers constantly explore the answers for a range of decision questions such as: Is the current form of the organisation appropriate to stay ahead of competition or economically viable? If not, *What* kind of changes are necessary to achieve organisational goals? *Where* to apply those change? and *When* to apply those changes?

Predicting answers to these decision questions requires precise understanding of organisational aspects, such as goals, organisational structure, operational processes, and its operating environment [14]. But analysing all relevant aspects and their dynamism is *exceedingly complex* [28] because of the inherent characteristics of the modern organisation that include socio-technical characteristics [24], complex and dynamic organisational structure [5], inherent uncertainty and emergent behaviour.

The state-of-the-practice of organisational decision-making chiefly relies on *qualitative* approaches, such as discussion and interviews, with limited *quantita-*

tive assistance that comes from spreadsheets based data computation. The role of the expert is paramount and excessive dependency on human intuitions and interpretations compounded with inadequate quantitative analysis often results in a less effective decision. This is especially true when the context is complex and dynamic [25]. We argues a suitable *quantitative* approach in addition to the *qualitative* and expert based approaches is a significant value add for modern organisations.

A range of enterprise modelling and analysis techniques supporting quantitative approaches exist. However, their utility is limited to a class of decision-making as compared to a wide range of decision-making discussed in management literature [5, 14]. For example, *inferential techniques* that rely on the statistical interpretation of historical system data are suitable only for static environments (*i.e.*, the environmental and organisational topology are fairly static with the time). The mathematical models, such as linear programming [12], work well for mechanistic and monolithic systems that are not adaptive in nature. The enterprise models, such as ArchiMate [19, 23], i* [30], and BPMN [29], are found to be inappropriate for the systems that exhibit significant uncertainty and emergentism [9]. Whereas the actor technologies [1] and agent-based systems [22] fall short of expressing the complex organisational structure and uncertainty (*see* Chapter 4 in [7]).

We aim to advance the state-of-the-art enterprise modelling and analysis technique to support quantitative evidence-driven decision-making. This paper focuses on two critical aspects of organisational decision-making - (a) *what* and *how* to capture the necessary information of an organisational decision-making, and (b) *how* to analyse various decision alternatives and understand their consequences prior to their implementation in reality.

We discern the necessary information of an organisational decision-making (*i.e. what* to capture) by reflecting on organisational theory [4] and management literature on decision-making [28]. Here, we present a novel approach to (a) effectively capture the necessary information of organisational decision-making using a Domain Specific Language (DSL), termed **OrgML**, and (b) analyse what-if scenario. The proposed OrgML combines two concepts: *system of systems* [11] and *actor model of computation* [1]. Our analysis approach draws upon a bottom-up simulation technique to understand the key characteristics of organisation such as: autonomy, adaptability, uncertainty and emergent behaviour.

2 Problem Space - Organisational Decision Making

Management theories [5] describe decision-making using three broad concepts, namely: *decision problem*, *course of action* and *decision*. The *decision problem* is *organisational goals* that an organisation targets, *courses of action* is the *knowledge of alternatives* that are considered and evaluated in a decision-making action, and a *decision* is the outcome of a decision-making action, *i.e.*, selected alternative. The literature also considers that a decision-making cannot happen in vacuum.

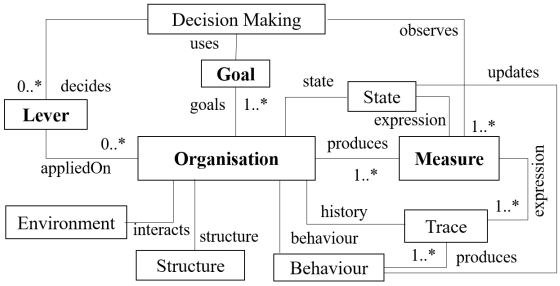


Fig. 1: Decision-making meta model

It requires specific *contextual information* to evaluate the consequences of potential courses of action, *i.e.*, develop *knowledge of consequences*. Methodologically, a decision-making is approached using four steps – (1) problem identification, *i.e.*, defining precise *decision problem* (2) generation of alternative *courses of action*, *i.e.*, development of *knowledge of alternatives* for a decision problem, (3) evaluation of courses of action or developing *knowledge of consequences* by predicting/computing the key performance indicators (KPIs) from *contextual information*, and (4) ranking of *courses of action* (*i.e.*, *consequent preference ordering*) and selection of the most effective course of action (*i.e.* a *decision*).

We represent the core concepts of organisational decision-making using a meta-model as shown in Fig. 1. The concept of decision-making are represented using three entities: *Goal*, *Measures*, and *Lever*. The concept *Goal* represents the organisational goals. *Measure* represents the *key performance indicators* (KPIs) that indicate the fulfillment of *Goals*. A *Lever* is a conceptual representation of a course of action. We refer these derived concepts as GM-L structure.

The contextual information for decision-making is represented using two primitive elements: *Organisation* and *Environment*. An *Organisation* is visualised as a *system* that has *Structure*, *Behaviour* and *State*. Moreover, an organisation often records its historical states, interactions, realisation of goals, and the useful phenomena as an *organisational memory* [21]. We term this historical records as *Trace*. Operationally, *Behaviour* updates the *State* and *Trace* of an organisation.

The concepts of GM-L structure and contextual information converge at two concepts: *Lever* and *Measure*. A *Lever* of a GM-L structure describes the changes of *Organisation* elements that include *Structure*, *Behaviour*, and *Goal*. Whereas, the*Measures* are expression over *Trace* and *State*. In this formulation, an organisational decision making is a method to develop the *knowledge of consequences* by computing/predicting the *Measures* for all identified *Levers* (*i.e.* the *knowledge of alternatives*), rank the *Levers* based on the observed *Measure* values (*i.e.*, *consequent preference ordering*), and select a *Lever* that serves the purpose best (*i.e.*, *decision*).

2.1 Characteristics and considerations

Management viewpoints perceive the characteristics of an organisation by reflecting on organisational theory [14] and system theories [18]. From organisational theory perspective, an organisation is a *reactive* entity (as it exchanges messages and resources with its environment). The *complexity theory* considers an organisation as a *complex* entity because it often composes a large number of interdependent subsystems or elements (*i.e.*, system of systems) in a *nonlinear*

way. Daft *et al.* further characterise an organisation as a composition of multiple *loosely coupled* and *autonomous* elements [14]. The *complex adaptive system* (CAS) theory [18] considers the behaviour of a complex organisation is largely probabilistic and emerges from the interactions of the connected sub-systems and individuals. Collectively they visualise an organisation as a complex *system of systems*, where each constituent system is characterised by multiple socio-technical properties such as: *modularity*, *composability*, *autonomy*, *temporality*, *reactiveness*, *adaptability*, *uncertainty* and *emergentism*.

From a methodology perspective, the modelling and analysis of necessary information witness a dilemma between the top-down versus bottom-up. In an organisation, the goal definition mostly follow a top-down path where the top level goals are decomposed into various unit level goals along the organisational structure. However, describing the overall behaviour of an organisation in the face of increasing complexity and uncertainty is a difficult proposition. The behaviour is known only for highly localised contexts, which suggest a bottom-up modelling approach. From an analysis perspective, the top-down (or reductionist) viewpoint helps to reduce the complexity but not able to recognise inherent emergentism, whereas the bottom-up analysis helps to understand the emergent behaviour. Therefore, a bottom-up analysis approach that has an ability to understand the emergentism is expected to be an effective *analysis aid*.

3 Proposed approach

An effective evidence-driven organisational decision-making depends on two factors: (i) an ability to capture all relevant decision-making information, such as *Goal*, *Measure*, *Lever* and contextual information as shown in Fig. 1, and (ii) an ability to perform quantitative *what-if* analyses. The former requires completeness and expressibility, and the latter expects the analysis efficacy.

To achieve our research goal and deal with the complexities presented in the previous section, we develop an actor-based domain specific language, termed as OrgML, to capture relevant information, and enable a bottom-up simulation. The key considerations for adopting *actor* abstraction are – *actors* are inherently modular, composable, autonomous and reactive entities. Therefore, it is capable of representing *system of systems* and the socio-technical characteristics of constituent systems (of a complex organisation). We extend the canonical form of actor abstraction to capture decision-making concepts: *Goal*, *Measure* and *Lever*. The primary reason for considering bottom-up simulation as an analysis aid are twofold: (a) it helps to observe the emergent behaviour of a complex system or system of systems, and (b) it can compute the behaviour of a system along time dimension by advancing the simulation 'time'. Therefore, the consequence of all hypothetical changes representing possible *Levers* along time dimension can be evaluated using simulation.

```
1  gml ::= GML {                                GML specification
2    goals    : (goal*)                        Goal specification
3    measures : (measure*)                     Measure Specification
4    levers   : (lever*)                       Lever specification
5  }
6  goal ::= id [ description ] g_expr           Goal declaration
7
8  g_expr ::= { g_expr g_reln g_expr }          Goal decomposition
9    | ⇒  leaf_goal                             Leaf level goal
10
11 g_reln ::= ';' | '|' | '→'                   And, or and sequence relations
12
13 leaf_goal ::= m_exp | r_exp                  Quantitative & relative expression
14
15 m_exp ::= measure                            Measure
16   |  integer | boolean | float | string      Constants
17   |  exp op exp                              Binary expression
18   |  Not exp                                 Negation
19   |  fun(exp*)                               Function call
20   |  [ exp* ]                                List of expressions
21
22 r_exp ::= [prefix] qualifier [suffix]        Relative expression
23
24 prefix ::= always | never
25 qualifier ::= increase | decrease            Relative operations
26   | maintain | maximise | minimise
27 suffix ::= t_exp time                        Time expression
28 t_exp ::= at | before | after | during
29
30 measure ::= id                               Measure declaration
31 lever ::= id                                 Lever declaration
```

Fig. 2: Syntax of GML specification

3.1 OrgML specification

A domain specific language and a supporting language workbench [16] is proposed to capture two sets of derived concepts, which we termed as: *GM-L structure* and *contextual information*. The GM-L specification language is designed to help decision makers of the organisation to specify GM-L structure using an intuitive and top-down manner. The organisation specification language that represents contextual information is designed for domain experts to capture necessary aspects and characteristics of an organisation in a bottom-up manner. The expressiveness is a key characteristic of organisation specification language. A seamless interoperability between two specification languages is established to ensure structural and conceptual consistency as they collectively specify the necessary information of an organisational decision-making.

GM-L specification : A concrete syntax of top-down GM-L specification is shown in Fig. 2. It contains *goals*, *measures* and *levers* specifications (line 1–5). A *goal* can either be decomposed into finer *goals* (as shown in line 8) or it can be mapped to a *measure* for a leaf level *goal* (as shown in line 9). The decomposition relationships (*i.e.*, *g_reln*) can be specified using one of the three goal decomposition relations: *and*, *or* and *sequence* (as shown in line 11). A leaf level *goal* to *measure* mapping can be specified either through a quantitative expression (*i.e.*, *m_exp*) or relative expression (*i.e.*, *r_exp*) as shown in line 13. The quantitative expressions are mathematical and logical operators over *measures* as shown in line 15–20, whereas the relative expression describes expected value of a *measure* with respect to its previous instances. A set of language constructs such as *increase*, *decrease*, *maintain*, *maximise* and *minimise* along with suitable

```
1  orgml ::= import_stmt calendar { element*}          OmgML specification
2  import_stmt ::= import ( orgml_spec_name* )         Import OrgML Spec
3  calendar::= Calendar   id { time* }                 Calendar entity
4
5  time ::= p_time                                     Primitive time
6      | time( integer ) of time                       Every nth occurrence
7      | time except time                              Not of time event
8      | [ time* ]                                     Sequence of time event
9      | anytime [time* ]                              A time from a list.
10
11 element ::= data_unit | org_unit                    Element types
12
13 data_unit ::= DataUnit   id  { ( variable*) }       DataUnit declaration
14
15 org_unit ::= OrgUnit id {                           OrgUnit declaration
16   goals: (goal*)                                    Goal specifications
17   variables:  (property*)                           Variables & traces
18   subscribes    : (time_event_name*)                Subscribed time events
19   consumes  : (event  [trace])*                     Incoming events
20   produces   :  (event  [trace])*                   Outgoing events
21   actions: (action*)                                Action specifications
22   measures: (measure*)                             Measure specifications
23   levers   :  (lever*)                              Lever specifications
24  }
25
26 property ::= [(@ indicator)] variable
27 variable::=  id :: type [ := exp ]                   Variable declaration
28
29 type::= element_id                                   DataUnit or OrgUnit
30  |  Integer | String | Double  |  Date | Boolean    Primitive types
31  |  [ type ]                                        List type
32
33 indicator::=  trace ( time_event_name)               Trace variable
34
35 event::= id (parameter* )                            Event definition
36 parameter ::= id type
37
38 action ::= on event  where condition do { stmt* }    Action specification
39
40 event ::= p_event                                    Primitive event
41      | time                                          Time event
42      | event[exp]                                    Number of occurrence
43      | no event                                      Event not occurred
44      | { event* }                                    Any event from list
45      | event between [event,event]                   Event between events
46      | [event*]                                      Sequence of events
47
48 p_event ::= id(type*)                                Event definitions
49
50 condition ::= {exp* }                                List of conditions
51
52 exp  ::= variable                                    Variable
53      |  integer | boolean | string | float | date    Constants
54      |  exp op exp                                   Binary expression
55      |  not exp                                      Negation
56      |  fun(exp*)                                    Function call
57      |  [ exp* ]                                     List of expressions
58
59 stmt  ::= variable := exp                            Assignment
60      |  new id(exp*)                                 Create new OrgUnit
61      |  p_event(exp) → id                            Send event
62      |  probably(exp) stmt else stmt                  Uncertainty
63      |  for (lvar:exp) do stmt                       Looping
64      |  if exp then stmt else stmt                   Conditional statement
```

Fig. 3: Organisation specification language syntax

prefix (such as *always* and *never*) and suffix are proposed to specify the relative expressions (as shown in line 22 and 24–28).

Constructs *measures* and *levers* are defined as labels as shown in line 30 and 31. They are expected to be introduced in GM-L specification and explicitly specified in organisation specification as part of interoperability.

Organisation specification : An organisation specification captures three initial key concepts – *OrgUnit*, *DataUnit* and *Calendar*. An *OrgUnit* is conceptually an *actor* that represents organisation, its constituent elements and environment in a modular form. A *DataUnit* is an abstraction to represent passive elements or data (where no behaviour is associated) of an organisation. The *Calendar* represents a set of meaningful 'time' events of an organisation. We recognise that future core concepts may emerge following experimentation and evaluation.

A concrete syntax of organisation specification is shown in Fig. 3. An organisation specification contains three sections: import, calendar and element description (shown in line 1). The import section `imports` a set of OrgML files that contain GM-L specification and other organisation specifications (enables modular specification as shown in line 2). The calendar section defines *time* as shown in line 3. A *time* definition can be two types – primitive time (underlying simulation engine decides this time interval) and composite time (can be computed from the existing time definitions). A syntax of time specification is presented in line 5 to 9. As shown, the time operators are – (a) n_{th} occurrence (*e.g.*, start of a week is every 7_{th} occurrence of day: `week = day(7)`, where day is a primitive time), (b) except (*e.g.*, a work schedule can be defined as all days except day 7 of a week: `schedule = day except [7]`), (c) a sequence of time events (*e.g.*, the second day of a week can be defined as `2ndDayOfWeek = [week, day, day]`), and (d) anytime is a probabilistic occurrence of a time event from a list of events.

Element description section defines *DataUnit* and *OrgUnit* as shown in line 11. A *DataUnit*, defined using term *data_unit*, contains a set of *variables*, where variables are typed elements as shown in line 13 and 27. A *type* can be one of the three alternatives – (i) primitive type, such as *Integer* and *String*, as shown in line 30, (ii) a list as shown in line 31, or (iii) an user defined type, such as *DataUnit* and *OrgUnit* as shown in line 29. An *OrgUnit* encapsulates its goals (optional), state, trace, event specification and (probabilistic) behaviour. The goal of an *OrgUnit* can be described using goal specification as shown in Fig. 2. The state variables that form the state of an *OrgUnit* can be specified using a set of *properties* (*i.e.*, typed variables) as shown in line 17, 26–27. Traces can be specified by augmenting a variable with '*trace*' keyword along with a time event (t) as shown in line 26 and 33. It implies that the marked variable will be recorder at every time interval (t) as *memory*.

An *OrgUnit* can be cognizant of time events when it subscribes them as shown in line 18. The subscribed time events helps an *OrgUnit* to exhibit temporal and autonomous behaviours. The interactions of *OrgUnits* are specified using events. An *OrgUnit* consumes a set of events, termed as incoming event, as shown in line 19, and produces a set of events, termed as outgoing events, as shown in line 20. These events can be traced (*i.e.*, the occurrence details will be recorded) when they are augmented with a keyword '*trace*' as shown in line 19 and 20.

The behaviour of an *OrgUnit* can be specified using a set of *actions* (line 21), where each *action* comprises a complex event specification (termed as *event*), conditional statement(s) (termed as *condition*), and behavioural specification as shown in line 38. The behavioural statement block (*i.e.*, {*stmt* *}) of an action it triggered when the complex event specification and conditional expression

```
1  measure::=exp@time_event display using chart_type
2
3  chart_type::= bar | pie | line | table          Chart Type
4
5  lever ::=      Lever     id (lever_spec* )       Lever declaration
6  lever_spec::= at event apply { lever_stmt* }     Lever specification
7  lever_stmt::= variable_name := exp               Variable assignment
8    | replace p_event By p_event                   Event replacement
9    | ignore p_event                               Ignore an event
10   | deactivate action                            Deactivate an action
11   | omit outgoing_event                          Don't send an outgoing event
```

Fig. 4: Syntax of Measure and Lever specification

over state variables are evaluated as true. A complex event can be specified using primitive events (*i.e.*, events which are raised from *OrgUnits*), time events (defined in calendar section), and complex operators on *events* as shown in line 40 – 46. The key operators are occurrence of an event (line 42), negation (line 43), any event from a list of events (line 44), an event between two other events (line 45), and sequence of events (line 46). A behavioural statement (*i.e.*, *stmt* of line 38) includes six types of statements (as shown in line 59): (i) variables assignment (line 59), (ii) creation of new *OrgUnit* (line 60), (iii) sending an event (line 61), (iv) probabilistic statement involving statements (line 62), (v) loop (line 63) and (vi) conditional statement (line 64).

As shown in line 22 and 23 in Fig. 3, a specification of an *OrgUnit* describes all *OrgUnit* specific *Measures* and *Levers*. A detailed syntax of measure specification is shown in lines 1–3 of Fig. 4. A measure is an expression over variables that needs to be observed at specific time interval using a suitable visualisation mechanism or *chart_type* as shown in line 3 (an extensible list of options). A *Lever* is a set of *lever_spec* (line 5) where each *lever_spec* is a tuple that contains an event and a collection of lever statements (*i.e.*, *lever_stmt*). A *lever_stmt* supports variable assignment, event replacement, ignore an incoming event, omit an outgoing event and deactivation of an action (as shown in line 7–11). To ensure the completeness of the overall specification, two rules are considered between GM-L specification and organisation specification – (i) Measure consistency: all measures of a GM-L specification should be owned by at-least one OrgUnit, and (ii) Lever consistency : all levers of a GM-L specification should be specified by at-least one OrgUnit.

Discussion: The concepts introduced in the proposed OrgML specification are grounded in well understood theories in the research literature. For example, the decomposition, modularisation and unit hierarchy of *OrgUnit* are taken from component abstraction [10]. An event driven architecture [26] is adopted to introduce reactive behaviour. The concept of intentional modelling [30] is adopted to enable goals. The complex event is traced to [27]. The goal-directed reactive and autonomous behaviour that may result into emergentism is traced to *actor model of computation* [1]. The visualisation scheme of the measures are taken from the visualisation of temporal data model presented in [2]. The time specification follows the notion of discrete and relative time definitions [17]. The lever specification is derived from variability modelling concepts [20]. Methodologically, OrgML supports a top-down approach for defining organisational goals and a bottom-up approach for behavioural specification.

3.2 Enabling bottom-up analysis - OrgML to Actor specification

We transform OrgML specification to actor specification for bottom-up simulation and *what-if* analysis. We consider an actor language, termed as Enterprise Simulation Language (ESL) [13], as the simulation specification. However, we define OrgML to actor specification mapping in such a way that any actor language, such as Akka [3] or Erlang [6], can be considered as simulation specification. Conceptually, all constituent elements of an organisation, namely: *OrgUnit*, *DataUnit* and *Calendar*, are mapped to a generic form of *actor*, which is represented in Fig. 5.

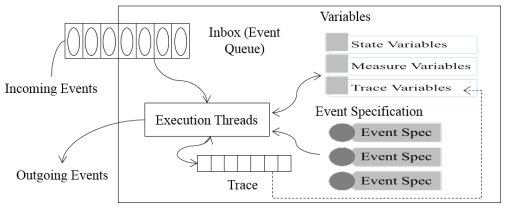


Fig. 5: Organisational decision-making

The OrgUnit *variables* are translated into actor variables, *traces* are translated into actor variables with list data-type, and the interactions among *OrgUnits* are mapped to event specifications. All incoming events and time events are considered as incoming event of an actor, *i.e.* they are queued into *inbox* of the recipient actor. The behavioural specification of OrgML *actions* are translated into event specification, where complex events are evaluated by maintaining an event trace (*i.e.*, history of events) and pattern matching algorithm. The statement to specify variable assignments, *new* actor, looping, conditional statement and *send* event are supported in most of the actor specification. Therefore, all statements can be mapped to actor specification by suitable syntactic transformation. All probabilistic statements are guarded with a conditional statement with a random number generation.

The *measures* are mapped to actor *variables*, which are recorded at specified time events using a list data structure and displayed using a visual graph.

In this formulation, a *DataUnit* is specialised actor that contains set of state variables. The *Calendar* is another type of specialised actor that contains a set of event specifications. A calendar actor receives primitive events (from underlying simulation engine), computes complex events, and sends complex time events to all subscribed *OrgUnits*.

3.3 What-if analysis

An approach for simulation driven *what-if* analysis is shown in Fig. 6. An OrgML specification that contains a set of *OrgUnits*, *DataUnits* and a *Calendar* is simulated for $time_s$ with or without a lever $lever_p$ to understand the as-is behaviour of an organisation or the consequence of a lever $lever_p$ over time $time_s$. For *what-if* analysis, the specification S of an organisation is first transformed into a new specification $S_{resolved}$ by applying a *Lever* specification $lever_p$ on S. The translated specification $S_{resolved}$ is then translated into an actor specification S_{actor} by applying OrgML to actor specification transformation rules. Finally, the *actors* of translated actor specification S_{actor} are executed in parallel using a simulation engine. Semantically, all translated *actors* concurrently process

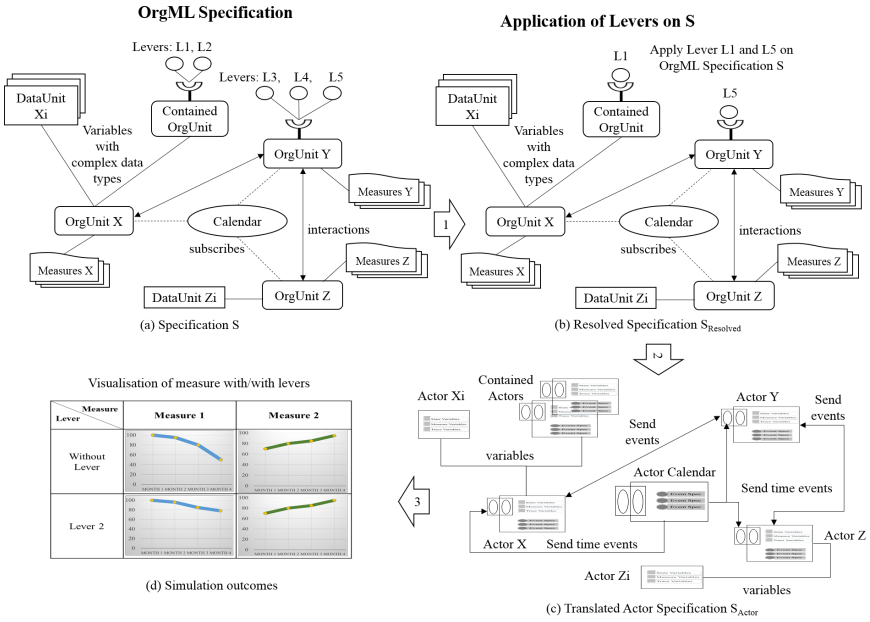


Fig. 6: Simulation of OrgML specification

events that include time events and incoming events from their respective *inbox*. Processing involves (a) dequeue events from its *inbox*, (b) update of trace information by appending processing event to its trace information, (c) evaluation of *action* applicability by evaluating the event trace and state condition of *action* specifications, and (d) simulation of behavioural specification of all valid *actions* that result into update of state variables, event interactions and creation of new actors. The processing of each time event computes relevant measure variables (of all *OrgUnits*) and displays using a specified visualisation format.

3.4 Implementation

A *language workbench* [16], termed as OrgML workbench, is implemented using Eclipse Xtext technology[1] to support standard language features [15]. The language features include three mandatory features: *notation*, *editor* and *semantics*, and three optional features: *validation*, *composability* and *testing*. In particular, the OrgML workbench supports a text-based *notation* (As shown in section 3.1 and illustrated in section 4), a transformational *semantics* using OrgML to ESL transformation (as discussed in section 3.2), a *free-form* eclipse-based *editor* with *syntax highlighting*, *folding* and *outline* features. The OrgML editors (*i.e.*, GM-L editor and organisation specification editor) support semantic services that include *reference resolution*, *error marking* and *live translation* of valid OrgML specification to ESL specification. In addition, it supports structural *validations*, *type checking*, and a *language unification* based language composability between GM-L specification and organisation specification.

[1] http://www.eclipse.org/Xtext/

```
1  GML ABCUniversity {
2   goals:
3     Goal ImproveRanking [improve University Ranking]  {
4        ImproveResearchQuality ; ImproveTeachingQuality
5      };
6      Goal ImproveResearchQuality ⇒ [  PublicationCount > 100 ];
7      Goal ImproveTeachingQuality ⇒ [  always minimise StudentConcerns ];
8   measures:
9     Measure StudentConcerns;
10    Measure PublicationCount;
11  levers:
12     Lever ImproveAcademicStudentRatio;
13     Lever IncreaseTeachingPreparation;
14 };
```

Fig. 7: An illustration of GM-L structure

4 Evaluation through Illustration

We evaluate the expressibility of OrgML through an illustration using a decision making problem of a hypothetical university, which is referred as ABC University. Consider a simplified case where ABC University is aiming to improve its teaching and research ranking by exploring possible courses of action, such as: (i) academic and student ratio, (ii) balance between research and teaching academics, (iii) work priorities of the academics, (iv) appropriate timetabling, and (v) experience and academic records of the academics.

4.1 GM-L Specification

For an illustration, we consider the stated goal of ABC University to increase its ranking has two sub-goals: improve research quality and improve teaching quality. We further consider that the research quality is a function of yearly publication counts and the teaching quality is a function over student satisfaction index that can be computed/predicted using the number of student queries and complaints. A representation GM-L structure of ABC University is shown in Fig. 7. As shown in the figure, the root goal of ABC University is captured using '*ImproveRanking*', which is decomposed into two leaf level goals: '*ImproveResearchQuality*' and '*ImproveTeachingQuality*' using an '*and*' decomposition relationship. The leaf goal '*ImproveResearchQuality*' is mapped to '*PublicationCount*' measure using a quantitative expression (*i.e.*, '*PublicationCount*' should be more than 100) whereas leaf goal '*ImproveTeachingQuality*' is mapped to '*StudentConcerns*' using a relative expression (*i.e.* value of '*StudentConcerns*' always should be in decreasing order). The specification also introduces two levers: '*ImproveAcademicStudentRatio*' and '*IncreaseTeachingPreparation*' for illustration.

4.2 Organisation Specification

An illustrative example of organisation specification is depicted in Fig. 8. The specification imports a GM-L specification (line 2) and contains a *Calendar*, a *DataUnit*, and a subset of *Academic OrgUnit* of ABC University.

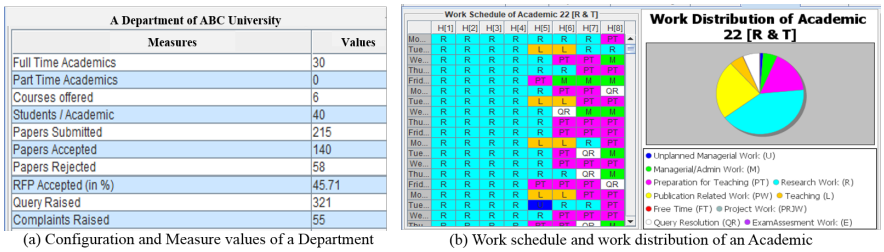
Calendar defines four time events, where *Hour* is associated to the '*primitive*' time; *Day* and Week are specified using expressions over time events; and *LectureSlot* is a complex time expression that specifies two slots in a week: second hour of Monday and fifth hour of Wednesday (shown in line 4).

```
1  Organisation Specification:: University {
2   import GML ABCUniversity;
3   Calendar    { Hour = Primitive;Day= Hour[8]; Week = Hour[40];
4      LectureSlot = [ Hour[2] Of Day[2] Of Week , Hour[5] Of Day[4] Of Week];}
5
6   DataUnit Module {
7      moduleName:: String := "Software Engineering";
8      credit::Integer := 5;
9   }
10 ...
11  OrgUnit Academic
12  {
13   variables:
14    academicName::String := "Academic1";
15    workingHours::Integer := 6;
16    @trace(Week) queryReceived::Integer := 0;
17    @trace(Week) compaintsReceived:: Integer := 0;
18    propensityOfTeachingPreparation::Integer:= 50;
19    teachingPreparation::Integer := 0;
20     ...
21   subscribes: Hour, Day, Week, LectureSlot; //Subscribe time event
22
23   consumes:
24    StudentQuery(Integer severity, Student student) ;
25    StudentComplaint(Integer severity, Student student);
26    PaperAcceptance(Integer paperId);
27   ...
28   produces:
29    Resolution(String resolution) ;
30    DeliverLecture(Module module, Integer hours);
31    PaperSubmission(Integer paperID);
32   ...
33   actions:
34    Action TeachingPreparation:  on [(Sequence [Day(x), Day(y)])  and (!
       LectureSlot(slot)) and (! StudentComplaint(severity2,student2))]
35    {
36       probably (propensityOfTeachingPreparation) {
37           teachingPreparation = teachingPreparation + 1;
38        };
39     }
40     ...
41   measures:
42    StudentConcerns=<queryReceived,compaintsReceived>@Week display using Line
43    PublicationsCount  = acceptedPaper  @Week display using Line
44 ...
45   levers:
46    Lever IncreaseTeachingPreparation : 'Increase Teaching propensity of
       academics from 50% to 80%'
47      apply [ propensityOfTeachingPreparation=80; ignore StudentComplaint; ];

48  }
49  OrgUnit Student { // Student Definition
50   ...}
51 }
```

Fig. 8: An illustration of organisation specification

A part of course *Module* is represented as *DataUnit* in line 6–9. It contains two typed variables with assignment expressions. A subset of *Academic OrgUnit* that illustrates *variables*, *subscribed* time events, *consume* events, *produce* events, *actions*, *measures* and *lever* definitions is shown in line 11–48. Defined *Academic OrgUnit* contains a set of variables where two variables are marked as trace variable with time event to specify *what* and *when* to capture them as trace (line 16 and 17). An action *TeachingPreparation* with a complex even condition is shown in line 34-39. It triggers in a day when no lecture slot is scheduled and no student complaint is received by the academic (line 34). It also illustrates a probabilistic behaviour in line 36. Two measure definitions with associated variables, time interval and visualisation are shown in line 41–43. Measure *StudentConcerns* is defined using two variables namely: query received and complaint received by



(a) Configuration and Measure values of a Department

(b) Work schedule and work distribution of an Academic

	Courses of action	Full Time Academics	Students Academic Ratio	Papers Submitted	Papers Accepted	Papers Rejected	Query Raised	Complains Raised
1	Initial Configuration (without lever)	30	40	215	140	58	321	55
2	Lever 1: Improve student/academic ratio	40	30	164	91	69	255	46
3	Lever 2: Increase teaching preparation	30	40	224	114	90	212	62

(c) Summary of what-if analysis

Fig. 9: A brief overview of simulation dashboard and what-if analysis

the academic. The specification indicate that these variables need to be captured at every week and displayed using line graph.

An illustrative *Lever* specification is shown in line 46–47. Lever *IncreaseTeachingPreparation* contains two lever statements – the first statement changes *propensityOfTeachingPreparation* from 50% to 80% and second statement *ignores* incoming event '*StudentComplaint*'.

4.3 Simulation and what-if analysis

The what-if analysis is performed using a complete OrgML specification of a department of ABC University with 30 academics and 1200 students[2]. First, the specification without any lever is translated to ESL specifications using OrgML to ESL translation rules, translated ESL specification is simulated for 52 *Weeks* (*i.e.* one year), and the specified measures are observed. An overview of the simulation dashboard is shown in Fig. 9. It shows the measure values of the department using a table (Fig. 9 (a)), work schedule of an academic using a table, and work distribution of an academic using a pie chart (Fig. 9 (b)). Subsequently, various *what-if* scenarios by applying levers to the initial configuration are explored. The outcomes of the what-if analyses are summarized in a table shown in Fig. 9 (c). The observations of these explorations (rows) help to understand the efficacy of the levers (with respect to specified goals) in a quantitative term and arrive at an informed decision.

5 Concluding remarks

Our key contribution in this paper is a novel domain specific language that is machine-interpretable and translates to simulation workbench to enable evidence-based informed organisation decision-making. The deeper analysis of the literature bought forth the core concepts, such as goal, measure and lever,

[2] The complete specification and simulation results of the case study can be found in Chapter 7.3 of [7]

and established the importance of socio-technical characteristics, such as *modularity*, *compositional*, *reactive*, *autonomous*, *intentional*, *uncertainty* and *temporal behaviour*, to precisely represent and comprehend a complex organisation. From a validation perspective, our focus is on the expressiveness of OrgML in the context of organisational decision-making, and the efficacy of its associated analysis capabilities. The key concepts of the language are validated through their derivation from current research literature of organisation decision-making. Sufficiency of expressive power of the OrgML language and analysis capability are demonstrated through an illustrative case study. In our research, we adopted design science methodology to develop and validated our research artifacts. We validated our contributions using a set of case studies. Our research methodology and other case studies are elaborated elsewhere [7]. Our validation establishes the *efficacy* and *utility* of OrgML and associated simulation capabilities. By implementing the language using established reference technology, we enable our research artifacts to become accessible for practitioners.

The key take away from our validation and usage in industrial context [8] are twofold - (a) considering simulation as a decision-making aid raises its own validity concerns particularly with respect to epistemic value of simulations, (b) while *efficacy*, *utility* and *completeness* of OrgML are established, the *usability* of OrgML needs to to improved. Exploring epistemological concerns raised through decision-making aids using simulation is our next focus area. A further potential area is the development of visual language notations to improve usability of our technology.

References

1. Agha, G.: Actors: A Model of Concurrent Computation in Distributed Systems. MIT Press, Cambridge, MA, USA (1986)
2. Aigner, W., Miksch, S., Müller, W., Schumann, H., Tominski, C.: Visualizing time-oriented data – a systematic view. Computers & Graphics **31**(3), 401–409 (2007)
3. Allen, J.: Effective Akka. O'Reilly Media, Inc. (2013)
4. Amagoh, F.: Perspectives on organizational change: systems and complexity theories. The Innovation Journal: The public sector innovation journal **13**(3), 1–14 (2008)
5. Anderson, D., Sweeney, D., Williams, T., Camm, J., Cochran, J.: An introduction to management science: quantitative approaches to decision making. Cengage Learning (2015)
6. Armstrong, J.: Erlang - a Survey of the Language and its Industrial Applications. In: In Proceedings of the symposium on industrial applications of Prolog (INAP). p. 8 (1996)
7. Barat, S.: Actor based behavioural simulation as an aid for organisational decision making. Ph.D. thesis, Middlesex University (2019), `eprints.mdx.ac.uk/26456/`
8. Barat, S., Khadilkar, H., Meisheri, H., Kulkarni, V., Baniwal, V., Kumar, P., Gajrani, M.: Actor based simulation for closed loop control of supply chain using reinforcement learning. In: International Conference on Autonomous Agents and MultiAgent Systems (AAMAS). pp. 1802–1804 (2019)
9. Barat, S., Kulkarni, V., Clark, T., Barn, B.: Enterprise modeling as a decision making aid: A systematic mapping study. In: IFIP Working Conference on The Practice of Enterprise Modeling. pp. 289–298. Springer (2016)

10. Barros, T., Ameur-Boulifa, R., Cansado, A., Henrio, L., Madelaine, E.: Behavioural models for distributed Fractal components. annals of telecommunications-annales des télécommunications **64**(1-2), 25–43 (2009)
11. Boardman, J., Sauser, B.: System of systems-the meaning of of. In: 2006 IEEE/SMC International Conference on System of Systems Engineering. pp. 6–pp. IEEE (2006)
12. Candes, E.J., Tao, T.: Decoding by linear programming. IEEE transactions on information theory **51**(12), 4203–4215 (2005)
13. Clark, T., Kulkarni, V., Barat, S., Barn, B.: ESL: An Actor-Based Platform for Developing Emergent Behaviour Organisation Simulations. In: International Conference on Practical Applications of Agents and Multi-Agent Systems. pp. 311–315. Springer (2017)
14. Daft, R.: Organization theory and design. Nelson Education (2012)
15. Erdweg, S., Van Der Storm, T., Völter, M., Boersma, M., Bosman, R., Cook, W.R., Gerritsen, A., Hulshout, A., Kelly, S., Loh, A., et al.: The state of the art in language workbenches. In: International Conference on Software Language Engineering. pp. 197–217. Springer (2013)
16. Fowler, M.: Domain-specific languages. Pearson Education (2010)
17. Goralwalla, I.A., Özsu, M.T., Szafron, D.: An object-oriented framework for temporal data models. In: Temporal Databases: Research and Practice, pp. 1–35. Springer (1998)
18. Holland, J.H.: Studying complex adaptive systems. Journal of Systems Science and Complexity **19**(1), 1–8 (2006)
19. Iacob, M., Jonkers, D.H., Lankhorst, M., Proper, E., Quartel, D.D.: ArchiMate 2.0 Specification: The Open Group. Van Haren Publishing (2012)
20. Kulkarni, V., Barat, S., Roychoudhury, S.: Towards business application product lines. In: International Conference on Model Driven Engineering Languages and Systems. pp. 285–301. Springer (2012)
21. Levitt, B., March, J.G.: Organizational learning. Annual review of sociology **14**(1), 319–338 (1988)
22. Macal, C.M., North, M.J.: Tutorial on agent-based modelling and simulation. Journal of simulation **4**(3), 151–162 (2010)
23. Manzur, L., Ulloa, J.M., Sánchez, M., Villalobos, J.: xarchimate: Enterprise architecture simulation, experimentation and analysis. simulation **91**(3), 276–301 (2015)
24. McDermott, T., Rouse, W., Goodman, S., Loper, M.: Multi-level modeling of complex socio-technical systems. Procedia Computer Science **16**, 1132–1141 (2013)
25. Meissner, P., Sibony, O., Wulf, T.: Are you ready to decide? McKinsey Quarterly, April **8** (2015)
26. Michelson, B.M.: Event-driven architecture overview. Patricia Seybold Group **2** (2006)
27. Paschke, A., Kozlenkov, A., Boley, H.: A homogeneous reaction rule language for complex event processing. arXiv preprint arXiv:1008.0823 (2010)
28. Simon, H.A.: The architecture of complexity. In: Facets of systems science, pp. 457–476. Springer (1991)
29. White, S.A.: BPMN modeling and reference guide: understanding and using BPMN. Future Strategies Inc. (2008)
30. Yu, E., Strohmaier, M., Deng, X.: Exploring intentional modeling and analysis for enterprise architecture. Enterprise Distributed Object Computing Conference Workshops (2006), doi=10.1109/EDOCW.2006.36