Model Based Enterprise Simulation and Analysis

A Pragmatic Approach Reducing the Burden on Experts

Vinay Kulkarni, Tony Clark, Souvik Barat, and Balbir Barn

{vinay.vkulkarni,souvik.barat}@tcs.com,
{t.n.clark,b.barn}@mdx.ac.uk

Abstract. Modern enterprises are complex systems operating in highly dynamic environments. The time to respond to the various change drivers is short and the cost of incorrect decisions is prohibitively high. Modern enterprises tend to exist in silos leading to fragmented knowledge with little support available for composing the fragments. Current practice places a heavy burden on experts by requiring a quick and comprehensive solution. This paper proposes a model based approach to this problem in terms of a language to be used for enterprise simulation and analysis that is capable of integrating the 'what', 'how' and 'why' aspects of an enterprise. Possible implementation is also hinted.

1 Introduction

Modern enterprises operate in a highly dynamic environment wherein changes due to a variety of external change drivers require a rapid response within a highly constrained setting. The cost of an erroneous response is prohibitively high and may possibly reduce options for subsequent changes in direction. Two further issues compound the problem. Firstly, the large size of modern enterprises means the understanding of 'what' is the enterprise, 'how' it operates and 'why' it so exists is available for highly localized parts only. Secondly, existing tool support addresses only one aspect, for instance, i* (http://www.cs.toronto.edu/km/istar/) addresses the 'why' aspect, BPMN tools (http://www.softwareag.com/corporate/products/aris/default.asp) address the 'how' aspect, ArchiMate (http://www.visual-paradigm.com/) addresses the 'what' aspect, etc. Moreover, these tools are not only non-interoperable but also paradigmatically different.

As a result, today, experts are forced to follow a process wherein: the problem under consideration is first decomposed into its 'why', 'what' and 'how' parts; these sub-problems are solved individually and independently making use of the available tool support to the extent possible; and the part-solutions are composed into a whole. This intellectually demanding endeavour becomes even more challenging due to the fractured knowledge and non-interoperable nature of current EA tools. The former constitutes the intrinsic complexity whereas the latter can be viewed as accidental complexity.

This paper explores whether model based engineering (MBE) can take some burden off experts' shoulders by reducing the intrinsic complexity. MBE has been used

M. Indulska and S. Purao (Eds.): ER Workshops 2014, LNCS 8823, pp. 3–12, 2014.

to good effect in systems development where models of systems are analysed before they are built. In some cases, parts of systems can be generated from models or run directly from models, thereby reducing the development and maintenance times.

Our proposal is that the application of MBE techniques to organizational decision making involves providing models that reflect the perspective of the key decision makers. To achieve this we apply techniques from Domain Specific Modelling (DSM) that engineers languages (DSLs) to contain concepts that are closely aligned with a given domain. In this way the key stakeholders engage with a support system that is business-facing. We construct an extensible core language that is used as the target of translations from a range of DSLs each supporting an organization analysis and simulation use-case. We then construct a virtual machine to support executed simulation for the core language. Thus, we envisage a system whereby an organization is modelled from a given viewpoint and transform the model so that what-if (*i.e.*, what will be the consequences of such and such action) and if-what (*i.e.*, what would have led to such and such situation) simulation can take place. Results of the simulation help determine choices between organizational change alternatives.

The rest of the paper is organized as follows: section 2 provides motivation in the light of current state, section 3 outlines the proposed solution rationale as well as the set of key features of enterprise specification language, section 4 describes ongoing work towards realization of the proposed solution, and section 5 concludes stating future research necessary.

2 Motivation

Key decision makers of an enterprise need knowledge about the current state of the enterprise, a set of change drivers, and a set of possible states (each of which is an improvement in terms of a well-defined evaluation criterion) so as to be able to make informed and preferably data driven decisions. Such a precise understanding of the current state of enterprise includes the understanding of 'what', 'how' and 'why' aspects of an enterprise and their interrelationships in order to provide answers to questions such as: *What optimization levers are still untapped? Will a change in business strategy percolate through to the IT systems? Will a particular change deliver the promised ROI? Which strategy is most likely to lead to a given desirable outcome?* Moreover, this understanding is required from multiple perspectives and at varying degrees of detail.

The current state-of-the-art of enterprise specification can be broadly classified as: those focusing on the 'what' and 'how' aspects [1,2,3] and those focusing on the 'why' aspects [4,5,6]. Supporting infrastructure for the former, with the exception of (http://www.visual-paradigm.com/) to an extent, is best seen as a means to create high level descriptions that are meant for human experts to interpret in the light of synthesis of their past experience. The stock-n-flow model [7] provides an altogether different paradigm for modelling the 'what' and 'how' aspects and comes with a rich simulation machinery for quantitative analysis (http://www.iseesystems.com/). Elsewhere, several BPMN tools providing simulation capability exist but are limited to the 'how'

aspect only (http://www.softwareag.com/corporate/products/aris/default.asp). Technology infrastructure for 'why' aspects (http://www.cs.toronto.edu/km/istar/) is comparatively more advanced in terms of automated support for analysis. However, correlating the 'what' and 'how' aspects of enterprise with the 'why' aspects still remains a challenge.

Given the wide variance in paradigms as well supporting infrastructure, the only recourse available is the use of a method to string together the relevant set of tools with the objective of answering the questions listed earlier. A lack of tool interoperability further exacerbates automated realization of the method in practice. As a result, enterprises continue to struggle in satisfactorily dealing with critical concerns such as business-IT alignment, IT systems rationalization, and enterprise transformation.

3 Proposed Solution

3.1 Rationale

From an external stakeholder perspective, the organization can be viewed as something that raises and responds to a set of events as it goes about achieving its stated objectives. The interface abstraction seems an appropriate fit to meet this perspective and can be extended with negotiation mechanisms such as specification of quality of service (QoS) and expectations from the environment – both of which are negotiation enablers. For instance, they allow definition of multilevel contracts, each promising to honour delivery of the stated objective at the specified QoS level iff the specified level of expectations from environment are met. An interface abstraction can only specify the 'what' and the 'why' of an organization. By providing an abstract implementation of an interface as a component the 'how' of an organization can also be specified. Such an (Interface, Component) tuple seems appropriate to abstractly capture an enterprise for the purpose of machine-based analysis.

Compared to the external stakeholders such as Customer, Regulating Authority *etc.*, internal stakeholders such as COO, CFO need views of the enterprise at a more granular level such as viewing the IT services organization as a set of interacting business units namely Sales & Marketing, Human Resources development, Software development, Quality control, Project management etc. Each of these business units have individual and independent objectives and can deploy different strategies to achieve them. Therefore, it seems the (Interface, Component) tuple can suffice to specify individual business units also. However, since these business units together constitute the organization, a [de]composition mechanism seems required. This requirement can be met if the component abstraction is first class as well as compositional.

We propose a modelling language engineering solution based on the principles of separation of concerns [8] and purposive meta-modelling. We posit a core language defined in terms of generic concepts such as *event*, *behaviour*, *property*, *interface*, *component*, *composition*, and *goal*. They constitute a minimal set of concepts necessary and sufficient for enterprise specification as argued below. The core language can be seen as a meta-model template where the generic concepts are placeholders. In the proposed approach a template emits the desired purposive meta-model

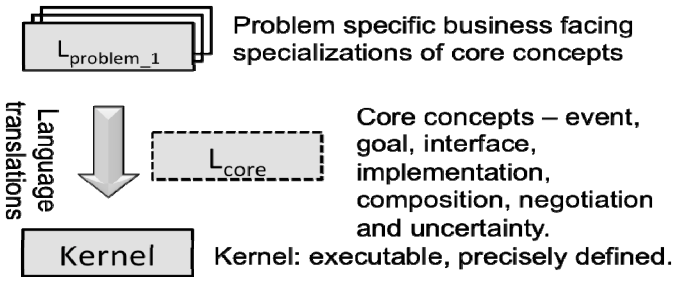

Fig. 1. Relation between Core, Kernel and DSLs

(i.e. a DSL) through a process of instantiation wherein the placeholder generic concepts are replaced by purpose-specific concepts. Our proposal is to construct an extensible kernel language that supports the same set of core concepts and supports both analysis and simulation. As purpose-specific DSLs and the kernel language share the same set of core concepts, the DSLs can be mapped onto the kernel language as shown in Figure 1. In other words, kernel language can be used as the target of translations from a range of DSLs. Each DSL supports an organization analysis and simulation use-case. We then aim to construct a virtual machine for the kernel language so that it is executable. Model execution supports organisation simulation and some analysis use-cases. Links to external packages such as model-checkers will complete the analysis use-cases.

The meta-modelling approach is suited to the open-ended problem space of enterprise modelling: any number of meta-models can be defined, relationships spanning across the various meta-models specified and the desired semantic meaning imparted etc.

Organisations consist of many autonomous components that are organized into dynamically changing hierarchical groups, operate concurrently, and manage goals that affect their behaviour. We aim for the kernel language to reflect these features by having an operational semantics based on the Actor Model of Computation (AMC) [9] and its relation to organisations, or *iOrgs* [10]. Our claim is that the AMC provides a suitable basis for execution and analysis of the core concepts and can be used to represent the features of a component. The key features that must be supported by the kernel language are detailed in [11].

4 Validation

Consider an organization that provides software development services. A client supplies a requirement for a system and expects to receive a completed system for an agreed price. It is in the interests of the service provider to deliver the system whilst minimizing costs and achieving a minimum QoS level.

The costs and QoS represent aspects of the business strategy for the organization. These can be decomposed into sub-goals that are eventually ascribed to various elements of the business such as individuals and departments. Such goals will contribute to the way in which the elements respond to events and requests that occur during the life-time of the organization.

An organization will naturally decompose into elements that correspond to physical or logical aspects of the business such as individuals, departments or IT components. Many of these elements will be autonomous, for example an individual will not wait to be instructed to perform a task, but may, at any time, decide on a course of action. Indeed, such autonomous behaviour can have important ramifications for the success of the overall business strategy. An individual who is highly motivated can proactively perform tasks that pre-empt future requirements. In addition, an individual whose goals are not aligned to those of the organization can take actions that are in their own interests and that are inconsistent with those that are imposed on them. In the case of the software services organization, an individual who is repeatedly overlooked as team-leader, may start to delay the delivery of software components.

Decision making and negotiation is an important part of implementing complex tasks within any organization. Individuals will make decisions that are based on their local knowledge and beliefs regarding the current situation. For example, without further direction, an individual may use a particular language to specify a software component because they believe it to be effective based on previous experience. Negotiation occurs whenever resources are limited, a programmer may need to negotiate regarding the availability of computing resources, and a manager may negotiate to take designers from one department to another for the duration of a project. The success of negotiation will depend upon a variety of factors, not least the local knowledge of the individuals that are participating.

In addition to long-lived groups that correspond to organizational components that might be seen on a conventional organogram, the life-time of a group may be much shorter; for example, a group of software designers that are convened and subsequently dissolved at the start of a project. Such groups need not have a physical realization, for example, operating via electronic communication mechanisms, however they have collective knowledge, may negotiate collectively and respond to events and requests as a group.

A senior decision maker may be interested in modelling the software service provider from a variety of perspectives. In all cases they are interested in minimizing costs and achieving a given minimum QoS. Perspectives include:

- Varying the number of resources available, for example changing the number of designers (who are relatively expensive) or the number of programmers.
- Imposing certain business directives such as a requirement that agile development methods should be used compared to traditional waterfall.
- Varying the abilities of individual roles within the organization with the resulting impact on the costs of development and QoS.
- Varying the roles within an organization. A specific role may be defined that is responsible for identifying opportunities for sharing good practice between different projects. Simulations can be run both with and without individuals responsible for the new role.

Given the requirements outlined above, our proposal is that the concepts can be represented using a fixed collection of concepts that include components, interfaces, events, goals and behaviour. These concepts can be implemented using features from the Actor model of computation and Multi-Agent Systems.

```
act resource_manager(time,resources,queue) {
  advance_time(t,d) =
    case (t+d) <= time {
      true -> time
      else t + d
    }
  Request(t,filter,action) ->
    case t <= time {
      true ->
        let requested = filter(resources)
        in case requested {
             Fail ->
               become re-
source_manager(time,resources,queue+[Request(t,filter,action)])
             Success(some_resources) -> {
               become resource_manager(time,resources-some_resources,queue);
               action(time,some_resources)
             }
           }
      else become resource_manager(time,resources,queue+[Request(t,filter,action)])
    }
  Release(t,some_resources,d,next) ->
    case queue {
      [] -> {
        become resource_manager(advance_time(t,d),resources+some_resources,[]);
        next(advance_time(t,d))
      }
      request:queue -> {
        become resource_manager(advance_time(t,d),resources+some_resources,queue);
        next(advance_time(t,d));
        send self request
      }
    }
}
```

Fig. 2. ESL Resource Manager

The kernel language is currently in development and is based on earlier work on the LEAP language and associated toolset [12,13,14]. The hypothesis of LEAP and the subsequent development of the ESL language is that, unlike current languages such as ArchiMate and KAOS, it is not necessary to provide a large diversity of modelling elements in order to capture the elements of interest when analysing aspects of organisations. LEAP proposes that components, ports and connectors can be used in conjunction with information models and behaviour rules in order to represent organisational features. However, this aspect alone is insufficient, and a key property of the LEAP and ESL languages is that they offer *higher-order features* including first-class components, functions and procedures.

Higher-order features provide a basis for abstraction over patterns of structure and behaviour that cannot otherwise be achieved by languages that are limited to first-order features. Abstraction is an important aspect of the EASE-Y approach because we aim to tailor the same architecture to multiple problem cases. Abstraction through higher-order features means that models can be parameterised with respect to different behaviours.

As an example of the use of higher-order features that implement a pattern of behaviour, consider the implementation of a resource manager to be used by the software service provider. In this case the resources are software engineers and the resource manager is to be used to simulate different strategies for implementing development projects.

```
act traveller(name,goals,plan,location,line) {
  Go(target) ->
    case plan {
      [] ->
        prove PlanJourney({location},{line},{target},plan) <- [],tube_planner {
          become traveller(name,goals,plan,location,line);
          send self Do
        } print('FAIL\n')
      else become traveller(name,goals+[Go(target)],plan,location,line)
    }
  Do ->
    case plan {
      [] ->
        case goals {
          g:gs -> {
            become traveller(name,gs,plan,location,line);
            send self g
          }
          else print(name + ' travels are complete.\n')
       }
      Move(_,location):plan -> {
        become traveller(name,goals,plan,location,line);
        print(name + ' moves to ' + location + '\n');
        send self Do
      }
      Change(_,_,line):plan -> {
        become traveller(name,goals,plan,location,line);
        print(name + ' changes line to ' + line + '\n');
        send self Do
      }
    }
}
```

Fig. 3. Planning

Figure 2 shows the implementation of a resource manager in the ESL kernel language. The resource manager is defined as an actor-behaviour. An actor is created with a behaviour that can be changed using the command **become**. The resource manager is parameterised with respect to the current time, the list of available resources and a queue of pending resource requests. The behaviour of an actor defines the interface of messages that the actor can process. Each message is handled in a separate thread of computation. In this case the resource manager can handle `Request` and `Release` messages. A request for resources includes the time at which the request is made, a resource filter and an action. The resource filter is a predicate that can be applied to the currently available resources to determine whether the request can be satisfied. The filter returns `Fail` when there are insufficient resources and returns `Success` otherwise. The action is supplied with the allocated resources. The higher-order features (filter and action) allow the resource manager to be used in a wide range of different resource allocation simulations. When the resources are released, the procedure `next` is used as a continuation.

In addition to rule-based execution, actors will be goal-driven and will need to plan. The ESL kernel language provides a deduction engine that can be used to develop plans in response to requests that are received as messages.

Figure 3 shows an outline of an underground train traveller actor that uses a collection of rules and a simple planner to construct plans when instructed to go to a target station. The ESL construct **`prove`** is used to construct a plan that is then enacted when the actor sends itself a `Do` message. Actions within the plan are either `Move` between stations or `Change` train lines. Figure 4 shows a general purpose

```
planner = rules {
  Solve(state,goal,plan,plan)  :- Subset(goal,state).
  Solve(state,goal,sofar,plan) :-
    Action(action,precons,add,delete),
    Subset(precons,state),
    \+ Member(action,sofar),
    DeleteAll(state,delete,remainder),
    Append(add,remainder,newState),
    Append(sofar,[action],sofar'),
    Solve(newState,goal,sofar',plan)
}
```

Fig. 4. A General Purpose Planner in ESL

STRIPS-like planner implemented in the ESL kernel. The language provides PROLOG-like deduction rules that are integrated with the actors and the associated information structures. The planner is used in the definition of a rule-set called `tube_planner` used in Figure 3. It is not envisaged that the senior decision makers will need to understand these concepts and features because the intention is to use techniques from language-engineering and model-driven development to offer a business-facing interface for each of the use-cases. For example, a decision maker in the software services example may be offered a simple interface that allows them to vary number of elements in a pre-populated model of their organization. Figure 5 shows the proposed EASE-Y architecture and the various stakeholders that take part in the process of developing a language for a particular use-case, deploying the language on the architecture and then using it to model part of an organisation, populate various concrete scenarios and perform simulation and analysis.

It remains a research question as to how much variability can be exposed in such a domain-specific way. Therefore, the proposed architecture for simulation and analysis will involve a number of stakeholders and involve a development process for each use-case. In the first instance the architecture will need to be tailored for a specific class of use-cases, for example languages may be developed that support software development companies. This will involve collaboration between a general domain expert and a language engineer and will result in a language definition that translated to a kernel using the general concepts. Secondly, the language will be tailored to the needs of a specific company involving collaboration between a company-specific domain expert and a language engineer. Thirdly, a domain specific language that is suitable for the decision maker will be produced. This will involve an expert in user-interface design. Finally the decision maker will be able to use the system unaided to configure various scenarios and run analysis and simulation. Of course, there are many opportunities within this process for reuse.

We modelled an IT services provisioning organisation in terms of i* (the ‘why’ aspect) and system dynamical (largely the ‘what’ aspect with little bit of ‘how’). We used i* and system dynamic models individually and independently to answer a set of questions. However, it was very hard to answer questions that need information of all the three aspects together. This was largely due to the paradigmatic differences between the two models and non-interoperable nature of the tools. We are in the process of specifying the same example using the proposed kernel language. We are confident of being able to answer all the questions herein.

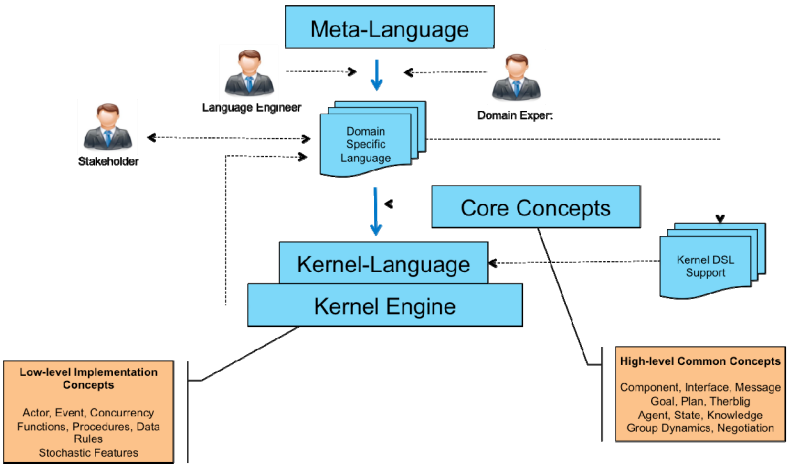

Fig. 5. EASE-Y Architecture

5 Conclusion and Future Work

Modern enterprises are complex systems operating in increasingly dynamic environment. Understanding of the 'what', 'how'' and 'why' aspects is essential for quick and comprehensive change response. We proposed a model based solution for reducing this inherent complexity and proposed technology infrastructure for automation. We believe this will take a significant burden off experts' shoulders. We hope to have an illustration of the proposed approach along with the supporting technology infrastructure ready very soon.

However, several challenges remain to be addressed pertaining to both inherent as well as accidental complexity. As regards the former, there is a need to support the inescapable realities of 'negotiation' and 'uncertainty'. Barring [15,16,17] very little work is reported. Paradigmatic differences and non-interoperable nature of existing EA modelling tools introduce accidental complexity which can only be addressed through a manual method. Automation support based on meta-model mapping and model transformation seems definitely possible. This can also help in maintaining the various models always in sync – a key unmet requirement that leads to eventual disuse of EA modelling tools. Size of modern enterprises and nature of problems they face results in very large enterprise models. Precision of analysis, speed of simulation, and life cycle support for these models that typically exist in a distributed manner are major hurdles to be overcome for the proposed approach to be useful and usable in real life. Many of the lessons learnt from use of MBE in application generation [18,19] seem readily applicable. Success depends principally upon the quality of models and only partially on the proposed technology infrastructure which is but a solution enabler. Ensuring semantic validity of enterprise models is another important challenge that needs to be addressed for the proposed approach to be usable in real life.

References

1. Josey, A.: Togaf v 9.1 enterprise edition - an introduction. The Open Group (November 2009)
2. Zachman, J.A.: A framework for information systems architecture. IBM Systems Journal 26(3), 276–292 (1987)
3. Wisnosky, D.E., Vogel, J.: Foo. In: Managing and Executing Projects to Build Enterprise Architectures Using the Department of Defense Architecture Framework, DoDAF (2004)
4. Dardenne, A., van Lamsweerde, A., Fickas, S.: Goal-directed requirements acquisition. Science of Computer Programming 20(1), 3–50 (1993)
5. Yu, E., Strohmaier, M., Deng, X.: Exploring intentional modeling and analysis for enterprise architecture. In: 10th IEEE International Enterprise Distributed Object Computing Conference Workshops, EDOCW 2006, p. 32. IEEE (2006)
6. Object Management Group. Business motivation model (bmm), version 1.1 (2010), `http://www.omg.org/spec/BMM/1.1/`
7. Meadows, D.H.: Thinking in systems: A primer. Chelsea Green Pub. (2008)
8. Tarr, P., Ossher, H., Harrison, W., Sutton, S.: N degrees of separation: multi-dimensional separation of concerns. In: Proceedings of the 21st Int. Conf. on Software Engineering, pp. 107–119 (1999)
9. Hewitt, C.: Actor model of computation: scalable robust information systems. arXiv:1008.1459 (2010)
10. Hewitt, C.: Norms and commitment for iorgs (tm) information systems: Direct logic (tm) and participatory grounding checking. arXiv:0906.2756 (2009)
11. Kulkarni, V., Clark, T., Barn, B.: A Component Abstraction for Localized, Composable, Machine Manipulable Enterprise Specification. In: 4th International Symposium on Business Modeling and Software Design (2014)
12. Clark, T., Barn, B.: Goal driven architecture development using LEAP. Enterprise Modeling & Information Systems Architectures-An International Journal 8(1), 40–61 (2013)
13. Clark, T., Barn, B.S., Oussena, S.: LEAP: A precise lightweight framework for enterprise architecture. In: Proceedings of the 4th India Software Engineering Conference, pp. 85–94. ACM (2011)
14. Clark, T., Barn, B.S., Oussena, S.: A method for enterprise architecture alignment. In: Proper, E., Gaaloul, K., Harmsen, F., Wrycza, S. (eds.) PRET 2012. LNBIP, vol. 120, pp. 48–76. Springer, Heidelberg (2012)
15. Jennings, N.R., Faratin, P., Lomuscio, A.R., Parsons, S., Wooldridge, M.J., Sierra, C.: Automated negotiation: prospects, methods and challenges. Group Decision and Negotiation 10(2), 199–215 (2001)
16. Bartolini, C., Preist, C., Jennings, N.R.: A software framework for automated negotiation. In: Choren, R., Garcia, A., Lucena, C., Romanovsky, A. (eds.) SELMAS 2004. LNCS, vol. 3390, pp. 213–235. Springer, Heidelberg (2005)
17. Olfati-Saber, R., Fax, J.A., Murray, R.M.: Consensus and cooperation in networked multi-agent systems. Proceedings of the IEEE 95(1), 215–233 (2007)
18. Kulkarni, V., Reddy, S., Rajbhoj, A.: Scaling up model driven engineering – experience and lessons learnt. In: Petriu, D.C., Rouquette, N., Haugen, Ø. (eds.) MODELS 2010, Part II. LNCS, vol. 6395, pp. 331–345. Springer, Heidelberg (2010)
19. Hutchinson, J., Whittle, J., Rouncefield, M., Kristoffersen, S.: Empirical assessment of MDE in industry. In: Proceedings of the 33rd International Conference on Software Engineering, pp. 471–480. ACM (May 2011)