Using Intentional and System Dynamics Modeling to Address *WHY*s in Enterprise Architecture

Sagar Sunkle, Suman Roychoudhury and Vinay Kulkarni
Tata Research Development and Design Center, Tata Consultancy Services, 54B, Industrial Estate, Hadapsar, Pune, 411013, India

Keywords: Enterprise Architecture, Intentional Modeling, System Dynamics, Decision Making.

Abstract: Taking and executing cost effective decisions in enterprises is becoming increasingly difficult due to multiple change drivers that affect varied aspects of enterprise. Enterprise architecture (EA) frameworks provide holistic treatment of ***what*s** and ***how*s** of enterprise but leave the important questions of ***why*s** unaddressed. Intentional modeling and system dynamics modeling provide treatment of ***why*s** at a point in time and over time respectively. We propose an approach where both intentional and system dynamics models are used in conjunction with EA models for a more effective treatment of ***why*s** than provided by either. Initial results with a case study suggest that best of both worlds may be obtained with such combined treatment of ***why*s** in enterprise.

1 INTRODUCTION

It is becoming increasingly difficult to take and execute cost effective decisions with regards to organizational changes in today's enterprises. Change, however small, to any entity in enterprise reverberates across varied dimensions like business, IT systems, and infrastructure (Kulkarni et al., 2013). Like other adopters we are exploring enterprise architecture (EA) to get a holistic view of enterprise (Sunkle et al., 2013a). We are also discovering that while EA enables creating reference models of the workings of enterprise, something more is needed to analyze these models to help cost effective decision making in enterprises. Apart from ***what*s** and ***how*s** of enterprise, mechanisms to explicate and analyze ***why*s** and ideally, the ability to play out alternative scenarios based on explicit ***why*s** are needed.

Two such mechanisms have been proposed earlier, tackling ***why*s** for EA with intentional modeling (using i* intentional modeling language) (Yu et al., 2006) and playing out EA scenarios using systems dynamics (Golnam et al., 2010). While abstractions in i* are close to abstractions in EA models, system dynamics is more generic in how it addresses dynamic evolution of behavior of system structures using just a few key abstractions namely, stocks, flows, and influencing variables (Sterman, 2000). Clearly, by combining strengths of i* and system dynamics, it seems possible to get more in terms of resolving ***why*s** than either can offer all by itself.

Instead of transforming EA models to system dynamics models as in (Golnam et al., 2010), we can begin with EA models, get i* models via metamodel mapping (Sunkle et al., 2013b), and then construct system dynamics models over i* models with the help of provided guidelines. This makes it possible to capture intentionality in addition to causality. This complete process begins with an EA model of enterprise and ends with an EA model which incorporates insights obtained by analysis over intentional and system dynamics models.

We illustrate this modeling and analysis process using evolution of our model-driven software development unit as a small case study. Instead of applying the process to evaluation of alternatives at a point in time using intentional modeling as in (Yu et al., 2006) or over time using system dynamics (Golnam et al., 2010), we demonstrate the utility of this process with the analysis of strategic *vulnerability* (Yu and Mylopoulos, 1994) over time. Early results show that this modeling process enables evolution of strategic intentions in enterprise over time.

The rest of the paper is organized as follows. Section 2 elaborates our motivation and presents key ideas of our approach. Section 3 describes the mapping between generic EA and intentional metamodels as well as interactive procedure used to create system

Sunkle S., Roychoudhury S. and Kulkarni V..
Using Intentional and System Dynamics Modeling to Address WHYs in Enterprise Architecture.
DOI: 10.5220/0004489200240031
In *Proceedings of the 8th International Joint Conference on Software Technologies* (ICSOFT-EA-2013), pages 24-31
ISBN: 978-989-8565-68-6

Table 1: Comparison of Various Models.

	Enterprise Architecture Models	System Dynamics Models	Intentional Models
1	Captures ***what*s** and ***how*s** of enterprise at a point in time	Captures ***why*s** in addition to ***what*s** and ***how*s** enterprise **over time**	Capture ***why*s** in addition to ***what*s** and ***how*s** of enterprise **at a point in time**
2	Descriptive	Prescriptive	
		Quantitative Assessment	Mainly Qualitative Assessment
3	Informative	Causal	Intentional
4	Highly Comprehensible	Low Comprehensibility	Highly Comprehensible
5	Low Simulation Capability	Very High Simulation Capability	Average Simulation Capability

dynamics models from i* models. In section 4 we reimagine our model-driven software development unit as an enterprise and apply combined i* and system dynamics modeling to ArchiMate-based model of this enterprise. Section 5 reviews related and further work and Section 6 concludes the paper.

2 MOTIVATION AND OUTLINE

EA is supposed to be the process of translating business vision and strategy into effective enterprise change. While EA models capture the working of the enterprise or the ***what*s** and the ***how*s**, the treatment of ***why*s**, even with the extended constructs for *motivation* in them, has been found to be more of *blueprint* nature (Wagter et al., 2012) and descriptive rather than prescriptive (van Bommel et al., 2007).

The prescription of *course of action* in the face of change is obtained using *causal* relationships in system dynamics models, whereas in intentional models it is obtained using *intentional* relationships as shown in Table 1. Causality denotes that there is a evolution relationship between two elements, i.e., if one entity changes over time then there will be change in the entity it is connected to. Intentionality on the other hand implies actor autonomy, i.e., it is up to an actor to deliver the responsibility that it is assigned to. Also, the nature of satisfaction of goals and soft goals in intentional models is qualitative; in the intentional modeling literature this is denoted by word *satisficed* rather than satisfied (Yu and Mylopoulos, 1994). Core elements in system dynamics such as stocks and flows are meant to deal with quantities (Sterman, 2000). Correspondingly, the course of action prescribed by intentional modeling and system dynamics is respectively qualitative and quantitative in nature.

Furthermore, EA models are specifically created for communication and shared understanding of ***what*s** and ***how*s**. They need to be highly comprehensible to be of any use at all. Intentional models also capture ***what*s** and ***how*s** that are relevant to the problem at hand in terms of strategic dependency diagrams and ***why*s** in terms of strategic rationale diagrams. System dynamics models on the other hand are created with simulation purpose in mind. As such, representation of problem, here in the context of EA, in terms of stocks and flows is often not meant for comprehension as much as for ease of simulation and scenario playing.

From the comparison in Table 1, it is clear that in order to apply intentional and system dynamics modeling for decision making in EA, it is better to treat EA models as *the* version of truth and use intentional and system dynamics models only as techniques to solve specific problems.

The process of modeling an EA problem and its solution using intentional and system dynamics models consists of 4 steps as enlisted below:

1. In the first step an intentional model is derived from the EA model using EA and i* metamodel mapping (Sunkle et al., 2013b).
2. In the second step, key questions pertaining to what stocks, flows, and variables in system dynamics models mean are used as described in Section 3.2 and a system dynamics model is manually arrived at from the intentional model.
3. In the third step, problem scenarios are played out and the results of dynamics analyses are transferred back to intentional models, by modifying original intentional models.
4. The changes in the intentional models are transferred back to original EA model. This revised EA model is actionable in the sense that it models the solution to the problem.

The next section begins with the description of EA and intentional metamodel mapping. Then the process of deriving system dynamics model from intentional model is described.

3 COMBINED TREATMENT

3.1 From EA to i* Models

We presented metamodel mapping of ArchiMate's ge-

neric metamodel (Group, 2012) with a metamodel of i* (López et al., 2011) in (Sunkle et al., 2013b). We map the active structure elements (ASEs) in ArchiMate metamodel such as business actors, application components, hardware and system software, as well as interfaces to actors in i*. The behavior elements (BEs) such as business processes, and business, application, and infrastructure services are mapped to tasks in i*. The passive structure elements (PSEs) are mapped to resources in i*, which are essentially informational or physical entities used or created by actors. This mapping enables the modeler to say that ASEs use or create PSEs while performing BEs as means to an end that is goal(s) and/or soft goal(s). For a detailed account of this mapping, reader is referred to (Sunkle et al., 2013b). Suffice it to say that with this bidirectional mapping it is possible to perform intentional what-if analyses on as-is EA model and use the best alternatives to construct the to-be EA model.

3.2 From i* to System Dynamics

***Why*s** are represented in i* using goals and soft goals. Goals are considered to be ends which can be achieved using tasks which may constitute alternative means. All elements of an i* model contribute to soft goals on varying level of positive or negative scale, *satisficing* them to varying extent. Essentially, both soft goals and goals may be considered stocks with rising or falling levels or values of satisfaction level.

If we treat goals and soft goals as stocks then rest of the elements in i* can be treated as variables that affect the flow of achievement and satisfaction levels of goals and soft goals. This process of mapping requires interaction of the modeler in the sense that questions pertaining to general nature of stocks, flows, and variables need to be asked of i* elements. We enlist the pointers which may be used in doing so below:

- In the SR model of an actor, the goals and soft goals are treated as stocks.
- For stocks that were goals in i*, a generic flow is to be considered and modeler should question what affects this flow. For instance, means-ends links in SR models indicate that tasks that are used as means to achieve a goal and clearly affect the flow of achievement as it were. In this case, the alternatives become variables.
- For stocks that were soft goals in i*, inflow and outflow captures satisfaction input and output. Modeler should question as to how these are affected from other elements that are stocks and flows in the system dynamics model being built up.

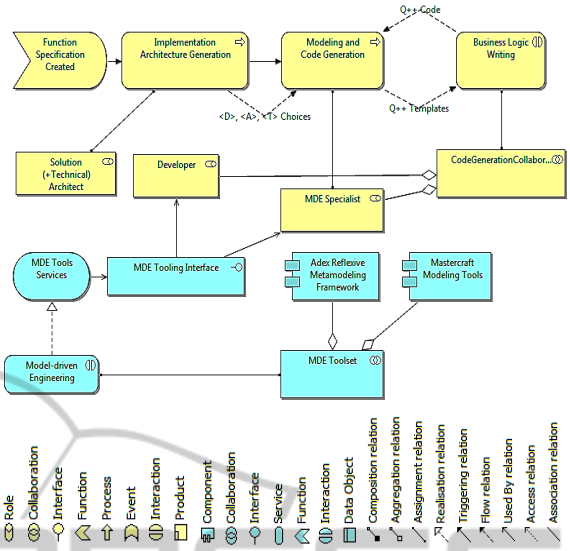

Figure 1: As-Is Enterprise Architecture Model of a Model-driven Software Development Unit.

- An actor's system dynamics model is to be encapsulated in some form of container available in the system dynamics modeling software being used. We are using iThink's evaluation version[1] to demonstrate current work. We use *module* construct in iThink which are self contained models that can be connected to other models. Each module in a given larger model defines a cohesive model of its own which fits our purpose.
- Strategic dependencies may be captured by module connectors. We use *assigned module inputs* in iThink which enable driving self contained modules to take input from and provide output to other modules.

It is presumed that actors involved in the problem with the as-is state of EA are already singled out in the i* models. Given that i* models look at ***why*s** qualitatively, we have chosen to represent satisfaction level as the entity to be quantified. If actual quantities of i* elements are already available they may be used as raising and lowering the satisfaction level in specific ways and these may be accommodated in the inflow and outflow equations. The above steps are repeated until all entities in SR and all dependencies in SD models are covered. At the end, a system dynamics model is obtained in which the ***why*s** that were at a point in time can now be played out over time.

The next section presents the case study and applies the complete process specified in Section 2 to the problem of addressing vulnerability of actors.

[1]We use evaluation version of system dynamics modeling software iThink http://www.iseesystems.com/softwares/Business/ithinkSoftware.aspx.

4 CASE STUDY

As the case study, we re-imagine our MDE-based software development unit as an enterprise. For the purposes of evaluation, we split the evolution of MDE-based software development into two stages and take them as as-is (before) and to-be (after) states.

The *first state* in our MDE-based software development unit was characterized by the use of a unified metamodel for specifying application, database, and GUI layers of an application, model-aware language for the developers to write business logic (Kulkarni et al., 2002), and separation of concerns in model-driven development for design strategies, architectural specifics, and technology platforms using aspects (Kulkarni and Reddy, 2003). The *second state* was characterized by the use of multi-user multi-site repositories for models and code with versioning and configuration management support (Kulkarni et al., 2010). The organizational change that prompted transition from the first state to second state was the *demand for onsite development of very large business applications*.

We capture the network of three roles namely, a solution architect (SA), an MDE specialist (MDESp), and a developer (D) in a given team. The current state of the enterprise is captured in the EA model of Figure 1. Using the mapping summarized in Section 3.1 and explained in detail in (Sunkle et al., 2013b), we obtain an i* model of the current state of enterprise explained above (Sunkle et al., 2013b, Figure 4).

As more and larger application requirements came to us, we had to change the nature of teams working initially on small applications to number of sub-teams performing several specialized tasks. These sub-teams needed to share a single main version of models and code in order to ensure that different models are consistent with each other and business logic is consistent with the models. The problem that we faced was *how to enable secure and synchronized access to models and code for large teams?*.

MDESp, D, as well as SA teams need to make use of models and code; MDESp needs access to models and metamodels, D needs to access code, and SA can search models and metamodels to make informed decision about choices of ⟨D⟩, ⟨A⟩, and ⟨T⟩ for current application. We use this setting to review concept of vulnerability which we analyze later using both intentional and system dynamics models of the problem stated above.

4.1 Analysis of Strategic Vulnerability

The concept of *vulnerability* was introduced in the context of intentional modeling in (Yu and Mylopoulos, 1994). The basic idea is that a depender may become vulnerable due to the failure of dependency. Of particular importance is mechanism of *assurance* of a commitment. Assurance means that there is some evidence that a dependee will deliver apart from dependee's claim. An assurance mechanism does not allow the depender to take action that can cause the dependee to correct its behavior. *Insurance* mechanisms on the other hand, reduce the vulnerability of depender by having more than one dependee for the same dependum. Apart from assurance and insurance of a commitment, it is possible to make it *enforceable*, only if depender and dependee are reciprocally dependent.

Intentional analysis of vulnerability (Yu, 2009) may be performed by classifying dependencies as open (failure does not affect depender), committed (depender is significantly affected), and critical (depender's goals may fail) when dependee does not deliver. Critical dependencies may result in severe vulnerability on depender's part.

We begin to investigate vulnerabilities of these actors with intentional model of the problem in the next section.

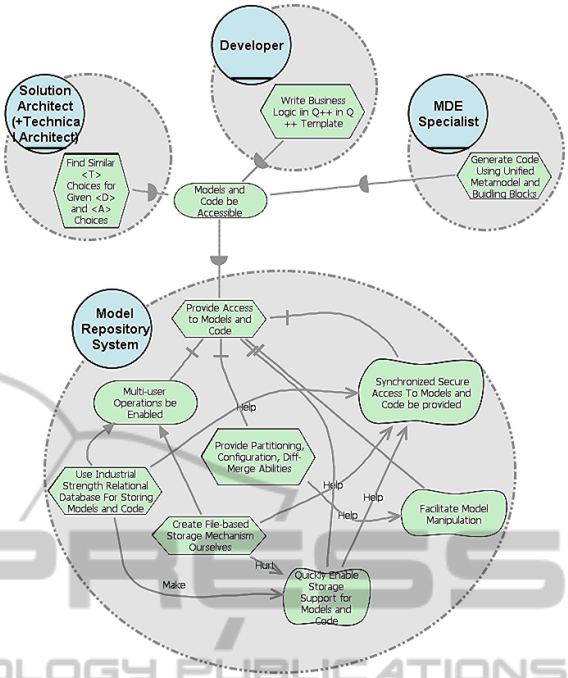

Figure 2: Enabling Secure and Synchronized Access to Models-Code for Large Teams.

4.2 Intentional Model

Evidently we needed to introduce a system whose

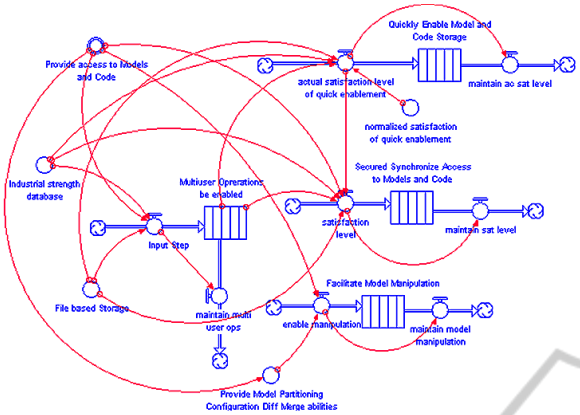

Figure 3: System Dynamics Model for Model Repository System's SR Model.

main task is to provide access to models and code for all these roles. Further requirement of this access are that *concurrent* multi-user operations should be enabled.

Choosing either alternative in Figure 2 leads to adding an actor in the form of a system, which is a (multi-user) model repository system (Kulkarni et al., 2010). It can also be seen that the responsibility of performing tasks that assure secure and synchronized access to models and code is assigned to the actor model repository system. The question with regards to vulnerabilities of SA, D, and MDESp is, *how would it make them vulnerable from the as-is state in which they were responsible for managing correct version of models and code themselves to the to-be state in which model repository system is responsible for the same*?.

From intentional model of Figure 2, it is clear that with complete dependence on model repository system for their respective core tasks, SA, D, and MDESp are critically dependent on model repository system. In other words, they are vulnerable in their dependence on the model repository system. The commitment is not enforceable since SA, D, MDESp and model repository system do not depend on each other for access to models and code. Yet, judgment about dependence in this case is qualitative and there is no way, due to the static nature of intentional models, to find out in what ways this vulnerability can be removed and whether to apply some sort of assurance mechanism or insurance mechanism. We could get better idea about the scenario in Figure 2 if it was played out. Next section describes how system dynamics modeling can be used for the same.

4.3 System Dynamics Model

We use the process described in Section 3.2 to manually arrive at system dynamics models from the intentional models. Since the vulnerability analysis concerns dependence of SA, D, and MDESp on model repository system, we begin with the SR model of model repository system shown in Figure 2 and we obtain system dynamics model shown in Figure 3. The goals and soft goals of Figure 2 are represented using stocks and tasks are represented using variables that control inflow and outflows. For instance, the goal *Multiuser Operations be enabled* and the soft goal *Secured Synchronized Access to Models and Code* are represented using stocks whereas the tasks are shown using variables. The inflows or outflows are programmed using a domain specific language (DSL) in iThink that supports event based (discrete or continuous) simulation.

Intentional model shown in Figure 2 also captures individual actors along with their dependencies. The system dynamics model shown in Figure 4 (Top) captures the notion of an actor in the form of a module, e.g., SA, D, MDESp, and model repository system are represented as modules. The directed arrows in Top of Figure 4 show how each actor is dependent on other actors. Note here that due to the modeling software's restrictions that modules can only communicate with modules, the mutual goal of *Models and Code Be Accessible* is also represented as a module. The directed arrow between *ModelRepositorySystem* and *Models and Code Be Accessible* depicts how the model repository system must provide access to models and code so that SA, D, and MDESp can do their tasks.

The system dynamics models of Figures 3 and Figure 4 (Top) help to analyze the vulnerability of system over time instead of at a point in time. A simple step function is applied to the variable *Provide access to Models and Code* in model repository system module that simulates the scenario where the model repository system cannot perform its core task during certain part of the day.

By playing out this scenario, we see that the underlying goals or soft goals of the model repository system such as *Facilitate Model manipulation* and *Multiuser Operation be enabled* fail resulting in non-delivery of required dependum to other actors. For instance, MDESp fails to perform its core tasks under such conditions since it is dependent on accessibility of models and code from the Model Repository System.

To actually see the evidence that using a backup support system can remove the vulnerabilities of SA, D, and MDESp, we create another module that represents such a system. Top of Figure 4 shows how this new system tries to arrest any failure arising due to in-

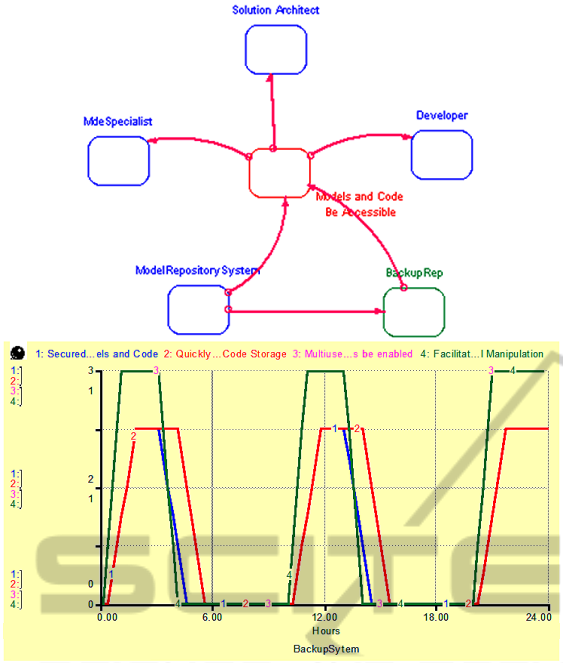

Figure 4: Top-Backup System as Insurance Mechanism against Vulnerability of Model Repository System, Bottom-Simulation of Backup System.

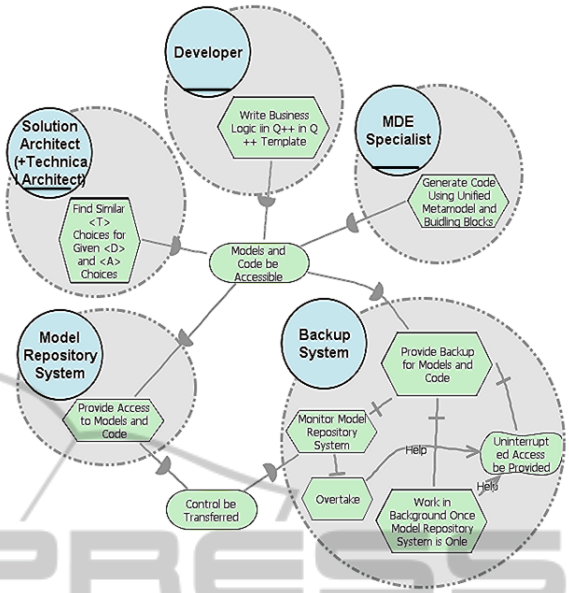

Figure 5: Intentional Model with Addition of Backup System.

accessibility of models and code in the model repository system. The backup system only takes over during those intervals when the main model repository system fails to operate.

The simulation of the backup system is shown at bottom of Figure 4 that clearly displays the intervals and duration when the backup system operates. With this new arrangement, the backup system and the model repository system become complimentary to each other. Consequently, actors whose reliance on the model repository system was critical can now perform their tasks without failure due to stoppage of work.

4.4 Getting Back To EA Model

The scenario played out using backup system indicates that it is beneficial to implement a backup system that takes over when model repository system is down. This is an insurance mechanism as we have introduced another dependee, backup system, for the delivery of the same dependum, (access to) models and code. With the evidence that substantiates choice of an insurance mechanism, it is time to reflect the same in the original intentional model of Figure 2. The process described in Section 3.2 is used to obtain the corresponding intentional model. Figure 5 shows addition of backup system in the original setting. SA, D, and MDESp are dependent on model repository system which is monitored by the backup system. Model repository system is dependent on backup system during downtime for the control to be transferred. At such time only SA, D, and MDESp become dependent on the backup system which provides access to models and code.

We can now use metamodel mapping in (Sunkle et al., 2013b) to get to EA model which captures the state that employs the solutions to the main problem of being able to support large development teams.

The EA model shown in Figure 6 builds on the model illustrated in Figure 1. Solution to the basic problem adds an actor, multi-user repository system. This is represented as an application component in Figure 6 via actor-ASE mapping. This application component is assigned to the application function that carries out model-code storage and model manipulation and realizes the application service which is used by SA, D, and MDESp via repository interface (shown as **(1)**). In the event of model repository system not working, the control is transferred to backup system which becomes responsible for providing SA, D, and MDESp with access to models and code (shown as **(2)**).

The newly constructed EA model is actionable in the sense that it can be used for clarifying the focus with regards to ASEs that are responsible to implement the solutions to problem(s) caused by the change, in what ways the solution impacts the way ASEs interact and how BEs to which they are assigned get affected in turn. Both target architectures of business and application layers are captured along

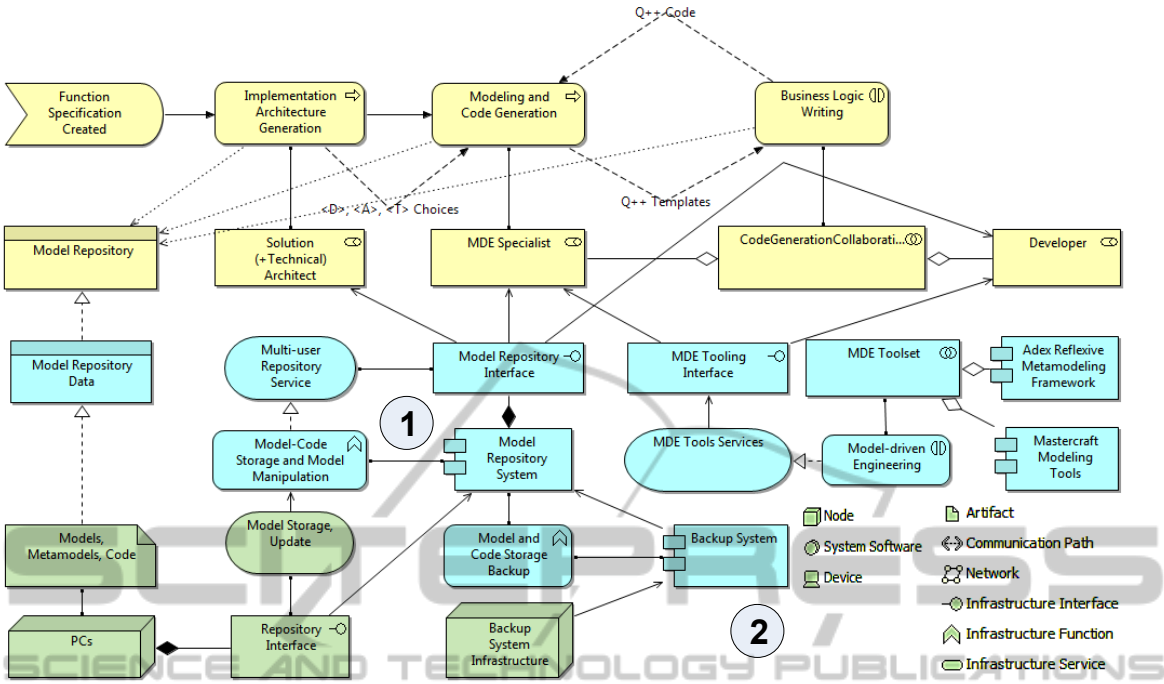

Figure 6: To-be EA Model of a Model-driven Software Development Unit. (For legends other than technology layer, please refer to Figure 1.)

with technology (hardware infrastructure) necessary to support these architectures as shown in Figure 6.

The way we have approached the combined usage of the EA, intentional, and system dynamics modeling languages is to restrict the scope of the problem to be solved from EA to i* and then from i* to system dynamics. With this arrangement, even though EA model is substantially large, latter models are created for specific problems to begin with. We suggest that modeler should scope the problem to be solved in terms of which actors' tasks and outputs are affected by given change and therefore should be involved in further modeling.

The total effect of using intentional and system dynamics models in concert is that cause-effect relationship between source and destination elements (from system dynamics models) may or may not take place depending on the intention of the owning actor (from the intentional models). Our approach brings together intentional and system dynamics models and initial results suggest that shortcomings of either approach mentioned above are addressed to some extent by their combined use.

5 RELATED AND FURTHER WORK

Intentional models lack sequential time based evolution of strategic rationale. They are also not good at reasoning about consequences of non-delivery of dependum. When a dependum is not delivered, the responsibility of serving the dependum to the depender may be delegated to another dependee. This delegation aspect is also less explored in intentional modeling, referred to as second and third level of actor autonomy (Du, 2007). System dynamics models on the other hand lack basic or first level of actor autonomy in the sense that freedom of actors in choosing alternative course of action cannot be represented out of the box.

The treatment of vulnerability aspect generally considers SD models as primary artifacts of analysis (Yu and Mylopoulos, 1994; Yu, 2009). We found in our experiments that the operationalization of the process of achieving the dependum captured in the SR model should also be included in the analysis since vulnerability arises from the possibility of non-delivery of dependum.

The combined usage of intentional and system dynamics models for treatment of ***why*s** suggested here is unique and to the best of our knowledge has not

been studied earlier in the context of enterprise modeling.

The core of our work is applying purpose-specific techniques (Sunkle et al., 2013b) to machine-manipulable and analyzable enterprise models (Sunkle et al., 2013a). Further work is therefore imagined in being able to express intentions and system dynamics not only in models of enterprise but also in models of actual systems and processes of enterprise and automating the bidirectional traceability between these.

6 CONCLUSIONS

Current enterprises face the daunting task of managing several aspects like business, IT systems, and infrastructure when responding to a change. While EA frameworks provide descriptive treatment of all these aspects, intentional and system dynamics models provide prescriptive treatment, suggesting courses of action so that desirable qualities are maintained as enterprise implements change. By combining intentional and system dynamics modeling and using them in the context of EA, we have shown that their respective shortcoming are addressed to some extent and a more rounded treatment of ***why*s** of enterprise is obtained. Starting with an EA model of the as-is state of enterprise, an actionable to-be state model is obtained via intermediate intentional and system dynamics models. While scalability can be major concern, our experiments suggest that by scoping the modeling activity at each subsequent transition, it is possible to make use of decision making capabilities of both intentional and system dynamics models.

REFERENCES

Du, Y. (2007). Incorporating system dynamics modeling into goal-oriented adaptive requirements engineering. In *Proceedings of the 2007 International Conference of the System Dynamics Society*, Boston, Massachusetts, USA.

Golnam, A., Van Ackere, A., and Wegmann, A. (2010). Integrating system dynamics and enterprise modeling to address dynamic and structural complexities of choice situations. In *Proceedings of The 28th International Conference of The System Dynamics Society*.

Group, O. (2012). *ArchiMate 2. 0 Specification*. Van Haren Publishing Series. Bernan Assoc.

Kulkarni, V. and Reddy, S. (2003). Separation of concerns in model-driven development. *IEEE Software*, 20(5):64–69.

Kulkarni, V., Reddy, S., and Rajbhoj, A. (2010). Scaling up model driven engineering - experience and lessons learnt. In Petriu, D. C., Rouquette, N., and Haugen, Ø., editors, *MoDELS (2)*, volume 6395 of *Lecture Notes in Computer Science*, pages 331–345. Springer.

Kulkarni, V., Roychoudhury, S., Sunkle, S., Clark, T., and Barn, B. (2013). Modeling and enterprises - the past, the present, and the future. In *MODELSWARD'13*. Accepted.

Kulkarni, V., Venkatesh, R., and Reddy, S. (2002). Generating enterprise applications from models. In Bruel, J.-M. and Bellahsene, Z., editors, *Advances in Object-Oriented Information Systems*, volume 2426 of *Lecture Notes in Computer Science*, pages 309–315. Springer Berlin / Heidelberg.

López, L., Franch, X., and Marco, J. (2011). Making explicit some implicit i* language decisions. In Jeusfeld, M. A., Delcambre, L. M. L., and Ling, T. W., editors, *ER*, volume 6998 of *Lecture Notes in Computer Science*, pages 62–77. Springer.

Sterman, J. D. (2000). *Business Dynamics: Systems Thinking and Modeling for a Complex World*. Irwin/McGraw-Hill.

Sunkle, S., Kulkarni, V., and Roychoudhury, S. (2013a). Analyzable enterprise models using ontology. In *Proceedings of CAiSE Forum*. Accepted.

Sunkle, S., Kulkarni, V., and Roychoudhury, S. (2013b). Intentional modeling for problem solving in enterprise architecture. In *Proceedings of International Conference on Enterprise Information Systems (ICEIS)*. Accepted.

van Bommel, P., Buitenhuis, P., Hoppenbrouwers, S., and Proper, E. (2007). Architecture principles - a regulative perspective on enterprise architecture. In Reichert, M., Strecker, S., and Turowski, K., editors, *EMISA*, volume P-119 of *LNI*, pages 47–60. GI.

Wagter, R., Proper, E., and Witte, D. (2012). A practice-based framework for enterprise coherence. In Proper, E., Gaaloul, K., Harmsen, F., and Wrycza, S., editors, *PRET*, volume 120 of *Lecture Notes in Business Information Processing*, pages 77–95. Springer.

Yu, E. S. K. (2009). Social modeling and i*. In Borgida, A., Chaudhri, V. K., Giorgini, P., and Yu, E. S. K., editors, *Conceptual Modeling: Foundations and Applications*, volume 5600 of *Lecture Notes in Computer Science*, pages 99–121. Springer.

Yu, E. S. K. and Mylopoulos, J. (1994). Understanding "why" in software process modelling, analysis, and design. In Fadini, B., editor, *Proceedings of the 16th International Conference on Software Engineering*, pages 159–168, Sorrento, Italy. IEEE Computer Society Press.

Yu, E. S. K., Strohmaier, M., and Deng, X. (2006). Exploring intentional modeling and analysis for enterprise architecture. In *Tenth IEEE International Enterprise Distributed Object Computing Conference (EDOC) Workshops*, page 32.