Experimentation in Executable Enterprise Architecture Models

Laura Manzur, John Santa, Mario Sánchez, and Jorge Villalobos

Universidad de los Andes, Bogotá, Colombia
{lc.manzur441,j-santa,mar-san1,jvillalo}@uniandes.edu.co

Abstract. Enterprise Architecture (EA) is a multidimensional model-based approach which enables analysis and decision-making in organizations. Currently, most EA approaches produce inherently static models: they focus on structural qualities of the organizations and represent their state only in one specific point in time. Thus, these models are not suitable enough for analyzing dynamic and run-time features of the organizations. This paper aims to solve this situation by proposing a model-driven platform for EA modeling and simulation. The proposal includes the means to build executable EA models, define experiments over the models, run the experiments, observe their run-time behavior, and calculate indicator-based results to aid the decision-making process.

Keywords: Enterprise Architecture, Executable Models, Discrete Event Simulation.

1 Introduction

The core of Enterprise Architecture (EA) approaches are models and diagrammatic descriptions of systems and their environment [7]. The main goal of EA projects is building said models, ensuring that they are an adequate representation of the enterprise in a specific point in time, and to use the models to diagnose problems or to propose improvements.

Usually, EA projects use metamodels to guide the construction of EA models. These metamodels are selected or designed depending on the project's needs and they define the types of elements that can appear in the models, the relevant structural and behavioral properties of each one, and their possible relationships. For example, a metamodel can establish: *i*) that the sole entities supported are Processes, Applications and Infrastructure Services; *ii*) the structural and behavioral properties that modelers can provide about each kind of entity (e.g., application name and service cost (structural), application response time and application failure probability (behavioral)); *iii*) that processes and applications can be related (a process supported by an application), and that applications and infrastructure services can be related as well (an application depends on some services). Furthermore, some metamodels include *constraints*, which provide additional information about valid or desirable model structures. Constraints can be used to specify that the total cost of the services invoked in a process instance should be less than $10, or specify that no process should depend on more than three applications (desirable).

EA models are typically analyzed by human experts. They study the elements, relations, and properties to identify problems and to derive conclusions that support decision-making processes. These analyses are made both by hand (studying the models directly) and by using tools that interpret the information present in the models. Nevertheless, "an enterprise architecture [...] is a static description of the essential components of the enterprise and their interconnections. By itself, this static description does not provide enough information to analyze and understand the behaviors that a given enterprise architecture is capable of producing" [6]. This means that there is information about the enterprises, and especially about their behavior, that cannot be evidenced just by looking at the models. As a result, this information is ignored during the analysis processes.

Including dynamic elements in enterprise architecture models is not straightforward. On one hand, dynamic elements are typically absent from standardized metamodels, and thus they cannot appear in the models. On the other hand, analysis tools, which are based on said standardized metamodels, provide only the means to work with structural and static aspects. Thus, they would not be able to use information about dynamic features even if it would be present. Finally, dynamic elements introduce a whole new complexity level for the analysis process that is beyond the capabilities of some analysis techniques.

To address this situation and enable the modeling and analysis of dynamic features, we propose a *model-driven platform for simulation.* Simulation is the key element here: by means of simulation, it is possible to draw inferences concerning run-time characteristics of an enterprise [1]. Moreover, our proposal has some characteristics that differentiate it from previous simulation proposals in the EA domain. First of all, it does not use pre-defined metamodels to describe the EA models that are simulated. Instead, specialized metamodels can be defined by metamodelers depending on their own particular needs, and on the kind and complexity of the analysis that they wish to perform. Secondly, the platform supports arbitrarily complex behavior without requiring changes to the base platform (e.g., simulated applications can perform parallel operations, organized by priority queues, and with different response times depending on their complexity). Thirdly, simulation results are processed offline: runs are observed, events and intermediate results are traced, and metrics are calculated. Metrics are defined by EA analysts, and they correspond to the indicators needed for their analysis and decision-making processes. Finally, this simulation approach uses highly configurable scenarios, that can be reused across many experiments, making the platform very suitable for answering *what if* questions.

This paper is structured as follows. Section 2 introduces the simulation metamodel and EA metamodel, which are the foundation of the simulation platform, and section 3 explains how these are used to build simulation scenarios together with indicators. Section 4 discusses simulation experiments, and section 5 presents and ideal workflow to use the proposed platform, as well as the results of an experiment. Finally, section 6 discusses related work and section 7 concludes the paper.

2 Metamodels for EA Simulations

Our simulation approach is based on two kinds of metamodels. The first one, represented by a generic *Simulation Metamodel* (SMM), abstracts concepts common to every simulation project, as well as their execution logic. Figure 1 presents the elements of the SMM. The central element is *Controller*, which maintains a timeline consisting of discrete *Instants*. An *Instant* is a point in time in the future[1] on which *Future Happenings* are scheduled to occur. *Future Happenings* represent meaningful events generated by the execution of the simulation, such as the completion of an Activity, Task or Operation; or the failure of an Application. A happening is described by an action that will happen to a *Referenced Element*, which is any of those defined in the Simulation Model. A special type of happening we called *Future Stimulus* represents events defined as input to the simulation (e.g., a client initiating a business process). In this case, the referenced element must be an instance of an element type that inherits from the *Stimulable* type. Section 4 presents further detail about stimuli.

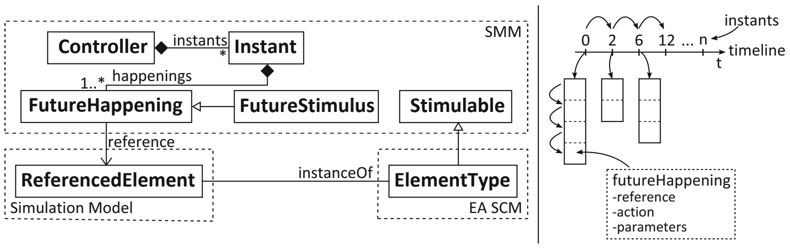

Fig. 1. Structure of SMM and timeline

During a simulation execution, the controller processes instants in chronological order. When all the happenings in an instant have been executed, the controller advances to the next one. To run the execution, we leverage on Cumbia [10][11]. Cumbia is a meta-modeling platform that supports the execution of models following the behavioral semantics specified in the metamodels, as well as complex monitoring requirements. In Cumbia, the elements of a metamodel are all *open objects*, which is a modeling abstraction formed by an *entity*, a *state machine*, and a set of *actions*. Since our proposal uses Cumbia as execution platform, the elements of the SMM must be described as open objects. For further information on Cumbia, refer to [11].

[1] The future is relative to the simulation execution. It refers to a point in time further from the current executing instant.

The second kind of metamodel in our approach defines the specific element types that will appear in the simulation models. On top of the typical characteristics of EA metamodels (attributes and relationships), these metamodels also include information about the behavior of each element type. There are certain concepts that appear only during the execution of a simulation such as '*process instance*', '*service invocation*', and '*current employee tasks*', which are not to be found in traditional EA metamodels. Concepts like these must be taken into account if we want the simulation to be as close to reality as possible, since they intervene in the operation of the real enterprise. This is critical in our approach, and requires this behavioral information to be detailed enough to allow the execution of the models. The advantage of this approach revolves around the ability to define behavior with arbitrary granularity and complexity, thus customizing the level of detail of the simulation. We call these metamodels, which are built on top of the SMM, *Simulation Capable Metamodels* (SCM).

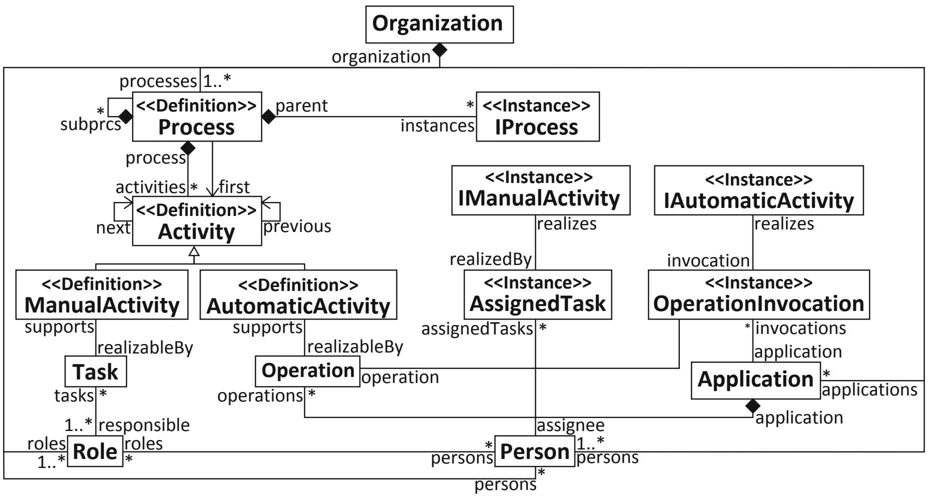

Fig. 2. EA SCM example

Figure 2 presents a simplified version of a SCM[2] from our metamodel repository. This SCM includes three EA domains: processes, applications and people (human resources). It models an *Organization* composed by a set of *Roles*, *Persons*, *Processes*, and *Applications*. Two types of activities are allowed: *ManualActivities* and *AutomaticActivities*, which are supported by *Tasks* and *Operations* respectively. Additionally, ⟨⟨*definitions*⟩⟩ are presented separately from ⟨⟨*instances*⟩⟩ because the behaviors of these kinds of elements are significantly

[2] Some relations between elements have been removed for legibility. The relations that present no multiplicity, correspond to 1. For a complete version of the metamodel, refer to [13].

different: Definitions are elements that provide a structure for guiding the creation of new instances; Instances are the elements that are executed and take time to be completed. This SCM also includes concepts and relations that appear only during the execution of the simulation, such as *invocations* (from Application to *OperationInvocation*) and *assignedTasks* (from Person to *AssignedTask*).

Because of the use of Cumbia as execution platform, the behavior of the elements is defined by state machines. These state machines define possible execution states for each element, and define the actions to perform during a simulation run. State machines are coordinated with events that trigger transitions; and with the invocation of methods implemented in the entities of the open objects.

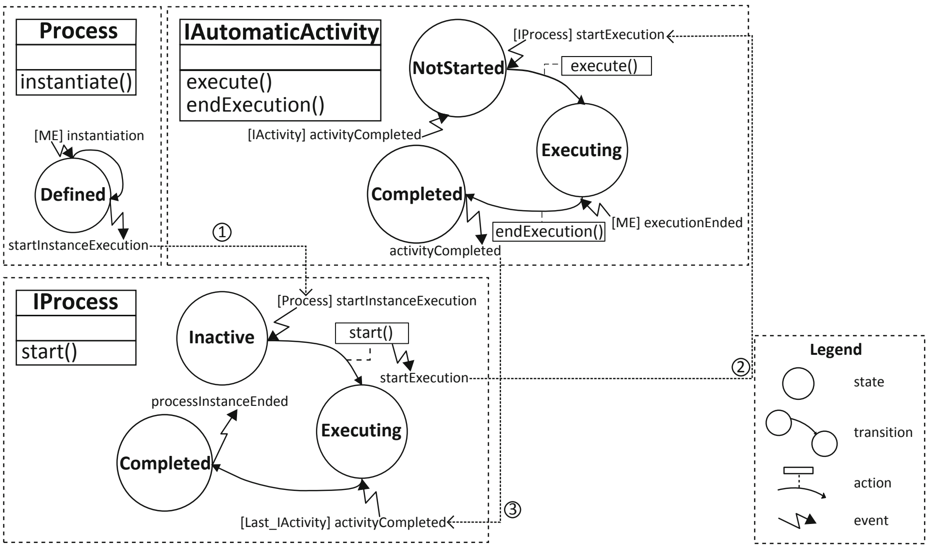

Fig. 3. State machines example

Figure 3 shows the open objects for elements Process, IProcess and IAutomaticActivity, and the coordination between them. It is necessary that every element of the SCM defines its corresponding state machine. Due to a lack of space, we will only explain the state machines from the mentioned open objects. A Process has only one state and an operation that creates an IProcess (`instantiate()`). IProcess and IAutomaticActivity include states that are coherent with the states of a process instance and an activity instance during their execution. When a process is instantiated during a simulation run, the Process element produces a `startInstanceExecution` event which is detected by the IProcess state machine (arrow 1), causing it to take the transition to the `Executing` state. This transition has a `start()` operation associated that produces a `startExecution` event. This event triggers the execution of the state machine of the first activity of the process. For this particular example, it is the IAutomaticActivity state machine (arrow 2). Once an activity has finished executing (transitions from `Executing` to `Completed`), it generates an `activityCompleted` event, which triggers the initiation of the next activity. Finally, when the last activity ends, it produces an `activityCompleted` event which notifies its parent IProcess (arrow 3), making the state machine take the transition to the `Completed` state.

The element IAutomaticActivity is a good example of the capabilities of the approach. Currently, the operation `execute()` finishes after an amount of time that depends on the current workload of the Application that realizes the operation. The calculation of the time is based on a probability distribution. The relations between element types of the architecture and how they behave can be easily modified by extending and adapting the metamodel [10].

To increase the reusability of the simulation models that conform to a given SCM, it is possible to set the values to any of its element's attributes from a configuration file that is external to the model. In this way, distinct configurations of the same simulation scenario can be tested without requiring new models for each case. We call this capacity parameterizability and models with this characteristic, *Parametric Simulation Models* (PSM).

3 Simulation Scenarios

A simulation scenario represents a simplified version of an organization which is interesting to be simulated, observed, and used to support decision making processes. Typically, one scenario is simulated several times with minor configuration differences to see which one offers the best outcomes. Moreover, to have comparable experiments, the characteristics observed in each one must be the same. To support this, a simulation scenario is composed by a PSM based on an SCM, and a set of *indicators* to group and present the results of simulation runs.

Figure 4 shows a PSM that conforms to the SCM described in section 2, which represents a small part of *Banco de los Alpes* (BDLA). BDLA is an EA scenario in the banking domain, which has been developed in our research group with the collaboration of representatives from different real banks, as a test case for several research initiatives related to EA [12].

The model represents the process for searching potential clients for the bank (*SearchNewClients*) which is composed of four activities: *LoadProspectsFromPartners*, *FilterUndesiredProspects*, *SegmentProspects* and *NotifyApprovedProductsToProspects*. These activities are respectively supported by four applications: an *FTP-ETL*, the *Clinton-MoneyLaunderingList*, a *CRM* and *ERP*. Notice that some element types do not appear in the PSM. This is because during the execution of this model, elements from concepts like *IProcess* and *OperationInvocation* will be created to represent instances of processes produced along the simulation execution and invocations to operations that support the process' activities. Therefore, elements of any type defined at the SCM with the Instance stereotype must not be defined in the PSM, but appear as the result of the simulation run.

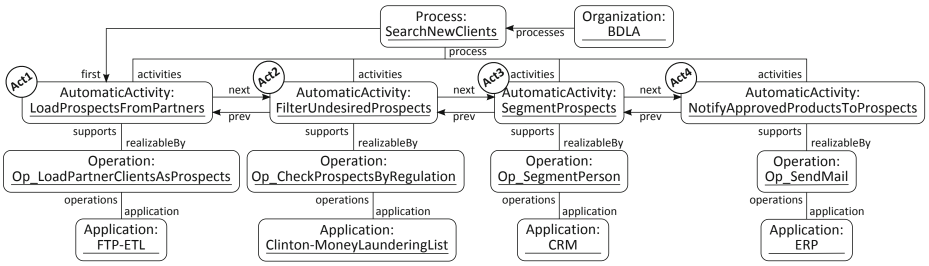
Process: SearchNewClients
Organization: BDLA
processes
process
first
activities
Act1
Act2
Act3
Act4
AutomaticActivity: LoadProspectsFromPartners
AutomaticActivity: FilterUndesiredProspects
AutomaticActivity: SegmentProspects
AutomaticActivity: NotifyApprovedProductsToProspects
next
prev
supports
realizableBy
Operation: Op_LoadPartnerClientsAsProspects
Operation: Op_CheckProspectsByRegulation
Operation: Op_SegmentPerson
Operation: Op_SendMail
operations
application
Application: FTP-ETL
Application: Clinton-MoneyLaunderingList
Application: CRM
Application: ERP

Fig. 4. PSM example

To complete a simulation scenario, a set of *Indicators* is defined to guide the collection of data about the simulation. As expressed by Frank in [5], an indicator refers to "quantitative measures that are specified using several mathematical constructs [...] for different types of reference objects and on various levels of abstractions in the enterprise". In our case, indicators refer to the specification of measurements to collect during the simulation. Indicators are described using an *Indicator Language*, which specifies how indicators should be presented and how to calculate them. Due to a lack of space to describe every aspect of our solution, we cannot give much detail about this language. Nevertheless, we will briefly present an example.

To calculate indicators, it is necessary to *observe* the simulation execution, by instrumenting the OpenObjects to gather relevant information. We took inspiration from the work in [9] to build an *Observation Structure*, composed by *Sensors* and *Tracers*, to collect data during the simulation execution. Sensors are in charge of monitoring elements in the simulation models, and detecting state changes in their state machines. When these occur, a sensor collects data and passes it to tracers. Tracers create *traces* with said data, and store them for offline processing. An observation model, which is automatically derived from an indicators definition, specifies the model elements to monitor, their events/actions of interest, the information that will be stored, and the corresponding tracers.

Figure 5 shows how this works. Firstly, a *Business Analyst* defines indicators (Is) by specifying (1) the desired visualizations of the simulation results, and (2) how to calculate them using variables (V). Additionally, he specifies the element types from the metamodel whose instances [from the model] (E) need to be monitored. The selection of these element types includes the places where sensors must be placed in the state machines. These places are called observation points. The sensors placed in these observation points are associated to a tracer (Trc) where traces (T) of intermediate states of the simulation are stored. After a simulation run is finalized, traces are processed and consolidated into indicators' representations, which are then presented to *Business Analysts* through the *Visualization Layer*. The indicator language's design is based in the metamodel proposed by Frank et al. in [5].

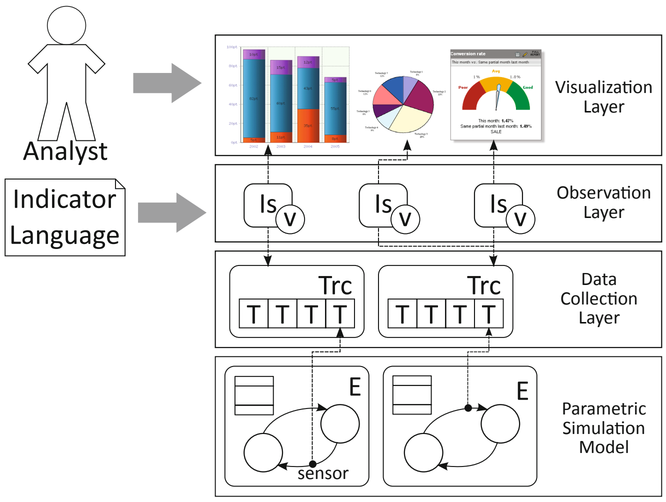

Fig. 5. Structure of observation model

```
 1  indicator-category Performance;
 2  specific-indicator StackedSearchNewClients {
 3    categories -> Performance;
 4    element ActivityInstance {
 5        type -> "IAutomaticActivity";
 6        filter -> "#self.process.parent.name == SearchNewClients";
 7        observation-points {
 8            timestamp initTime -> "execute";
 9            timestamp finishTime -> "endExecution";
10        }
11    }
12    variables {
13        double[ ] automaticProcessingTime -> "ActivityInstance.finishTime -
14        ActivityInstance.initTime" groupBy "ActivityInstance.process.name";
15        string processName -> "ActivityInstance.process.name" groupBy
16        "ActivityInstance.process.name";
17    }
18    visualization {
19        title -> "Stacked search new clients";
20        bar-chart {
21            orientation -> horizontal;
22            stacked -> true;
23            bars {
24                bar "processName" -> {"automaticProcessingTime"};
25            }
26        }
27    }
28  }
```

Listing 1.1. Example of indicators definition

Listing 1.1 presents an example of the Indicator Language. Line 1 presents the definition of an indicator category. These categories classify the desired indicators so they can be grouped according to the Analysts interests (line 3). To specify an indicator, an Analyst first specifies the elements to monitor (line 4 - 11). In the sample case, only instances of automatic activities are monitored (line 5). For each element, we also specify a filter (line 6) to guarantee that sensors are only placed in the automatic activities of the SearchNewClient process and not on activities from other processes. Then, we identify the observation points for the element which corresponds to actions in transitions in the element's state machine (lines 8 and 9). After the elements to monitor, an analyst then specifies the variables and formulas to calculate the indicator (lines 12 - 17). Finally, variables are used to build the indicators representation. Lines 20 to 24 show that for the Search New Clients process indicator the results will be presented using a stacked bar chart.

4 Simulation Experiment

Experimentation in simulation is the ability to configure and test distinct scenarios to observe the outcomes of each simulation execution, and to relate these results to the models used as input. This serves to answer *what if* questions about the operation of the organization. We refer to an experiment as a particular configuration of a scenario (PSM and indicators), complemented with the definition of stimuli that represent influences to the model.

To configure PSMs, we currently use property files that set the values for certain attributes of the model. The aspects that can be configured using these property files range from selecting a specific probability distribution with given parameters to describe the amount of time an Operation takes to complete, or more complex behavior such as specifying a task assignment policy (taken from a set of predefined policies). Listing 1.2 presents an example of a configuration for the PSM introduced in the previous section. Line 1 specifies the distribution to calculate the response time for the operation Op_LoadPartnerClientsAsProspects and lines 2 and 3 present the parameters to calculate said distribution. The rest of the file configures similar properties for the other operations that support activities in the process.

```
 1  Op_LoadPartnerClientsAsProspects.responseTime.distribution=Normal
 2  Op_LoadPartnerClientsAsProspects.responseTime.mu=120000
 3  Op_LoadPartnerClientsAsProspects.responseTime.sigma=20000
 4
 5  Op_CheckProspectByRegulation.responseTime.distribution=Triangular
 6  Op_CheckProspectByRegulation.responseTime.min=10000
 7  Op_CheckProspectByRegulation.responseTime.max=25000
 8  Op_CheckProspectByRegulation.responseTime.mode=20000
 9
10  Op_SegmentPerson.responseTime.distribution=Uniform
11  Op_SegmentPerson.responseTime.min=5000
12  Op_SegmentPerson.responseTime.max=8000
13
14  Op_SendMail.responseTime.distribution=Constant
15  Op_SendMail.responseTime.value=30000
```

Listing 1.2. Configuration example for elements of type Operation

On the other hand, a simulation experiment includes a *Stimuli* definition. Stimuli are influences over the model that affect some of the organization's elements at certain time intervals. Since an organization isn't an entity detached from its surroundings, people or events from its environment can affect its operation. For example, a possible stimulus for the BDLA scenario would be a client that initiates a process to open a product. Stimuli are defined using a *Stimuli Language*. Listing 1.3 shows a small example of stimuli for the PSM introduced in the previous section. The stimuli language allows defining who generates the stimulus (an element of the model typed *Stimulable* – e.g. a Person named `John`, line 2) and the action it triggers (e.g., `instantiateProcess`, line 4). Line 5 indicates that the action `instantiateProcess` requires specifying the name of the process to be started (`SearchNewClients`). Additionally, for each stimulus it is necessary to provide information about the time intervals it occurs. The first occurrence of the stimulus is defined by its initiation time, which can be at any specific time of the execution (e.g., `30.0 minutes` after the simulation began, line 9) or at the beginning (`0.0 milliseconds`). The following occurrences of the stimulus are inserted in the timeline according to a probabilistic distribution. Line 10 specifies that stimuli will be inserted using a Chi distribution with `5 degrees of freedom`. Finally, the termination condition indicates when to stop creating occurrences of the stimulus and can be specified as a certain time after the first stimulus is inserted in the timeline or as the maximum number of repetitions of the stimulus (e.g., `5 times`, line 16).

```
 1  stimulus {
 2      actor "John";
 3      action {
 4          name -> "instantiateProcess";
 5          parameters {
 6              string process -> "SearchNewClients";
 7          }
 8      }
 9      time -> 30.0 minutes;
10      distribution {
11          chi {
12              degrees-freedom -> 5;
13              square -> false;
14          }
15      }
16      termination-condition {
17          repeat-until -> 5 times;
18      }
19  }
```

Listing 1.3. Example of stimuli definition

Once a simulation experiment has been completely defined, it can be loaded in the *Simulation Engine*. This engine runs the simulation while sensors gather information about the execution and store it in traces. After executing the simulation, traces are processed and indicators are calculated and presented using a *Visualization Tool*. The results can then be evaluated by *Business Analysts*, which propose new experiments until a proper architecture that satisfies the organization's needs is found.

5 EA Simulation Workflow and Validation

The elements presented in the previous sections are the building blocks of this paper's proposal. Figure 6 presents an ideal workflow to build and execute a simulation. First, Business Analysts define the simulation requirements, taking into account their interests in the project. From these requirements, *EA Metamodelers* design (or select from a repository) an EA SCM that matches them using a *Metamodeling Tool*. Said SCM must extend from the SMM to include simulation concepts.

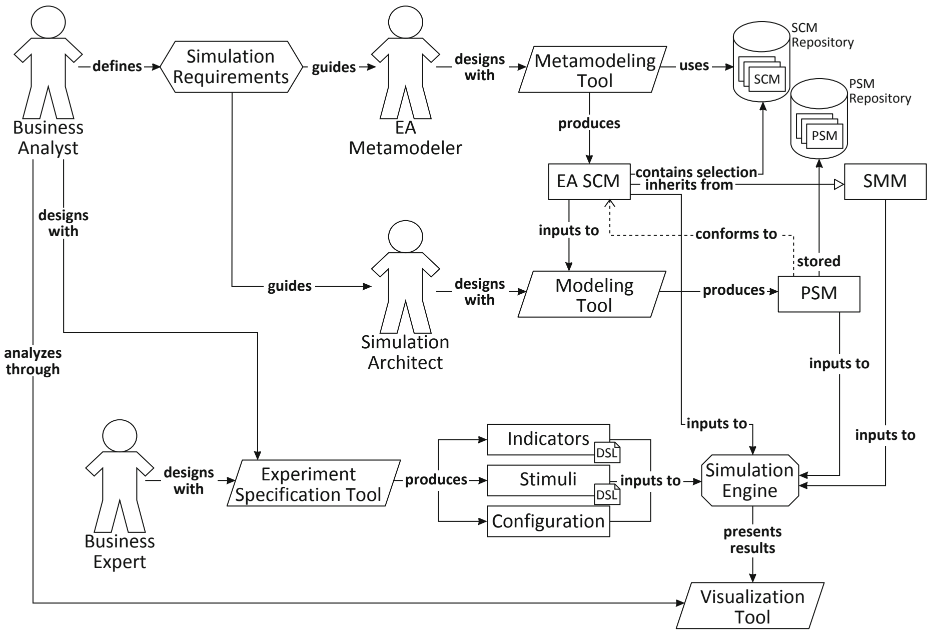

Fig. 6. A workflow for applying the proposed approach

After an SCM has been created, a *Simulation Architect*, guided by the requirements, builds a PSM using a *Modeling Tool*, which conforms to the EA SCM designed by the *EA Metamodeler*. Afterwards, Business Analysts use an *Experiment Specification Tool* to complete the definition of a simulation scenario by defining the desired indicators. With this same tool, *Business Experts* build experiments by specifying the model's Configuration and the Stimuli that will guide the simulation execution. Finally, all these artifacts are used as inputs for the *Simulation Engine*, which runs the experiments and presents the results through a *Visualization Tool* for Business Analysts to evaluate and make decisions accordingly.

As mentioned in section 3, we proposed a PSM using parts of a scenario called *Banco de los Alpes* (BDLA) [12]. BDLA is a banking initiative focused on the young adults market segment and on venture projects. It has approximately 25 processes distributed along 5 macroprocesses in its value chain. To validate our proposal, we use the following scenario/experiment, whose design cannot be explained due to a lack of space. We selected the SCM presented in figure 2 and a conformant PSM (figure 4), as well as a set of indicators (listing 1.1) that aim to studying the SearchNewClients process. For each run of the experiment, the engine created 3 instances of the process as specified in the stimuli definition (listing 1.3), which behaved according to the configuration presented in listing 1.2.

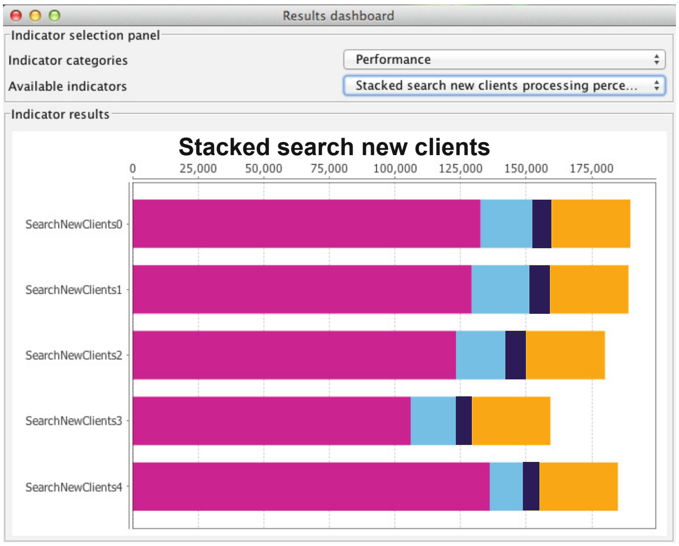

Fig. 7. Indicator of a simulation execution

The SearchNewClients process has solely automatic activities, but in our simulation they can take a significant amount of time to complete. Figure 7 presents the results of the defined experiment, where we can observe that the first activity the process executes (LoadProspectsFromPartners - Act1) corresponds to more than 65% of the completion time in every instance of the process. Table 1 shows the activities completion times in milliseconds. We can observe that the last activity of the process (NotifyApprovedProductsToProspects - Act4) has no effect over the differences in completion times of the instances since it always takes the same amount of time to complete (30,000 milliseconds). The average completion time of the process instances is 180,5756 milliseconds. We can observe that the process instances completion times ranged from 12% less than the average time up to 5% more. This happens because we used probabilistic distributions to calculate the response times of the process' activities, allowing the simulation to obtain different results in each experiment.

Table 1. Results of defined indicator in milliseconds

Process Instance	Act 1	Act 2	Act 3	Act 4	Total
SearchNewClients0	132,673	20,042	6,973	30,000	189,688
SearchNewClients1	129,314	22,081	7,596	30,000	188,991
SearchNewClients2	123,315	19,045	7,646	30,000	180,006
SearchNewClients3	105,987	17,371	5,883	30,000	159,241
SearchNewClients4	136,083	12,945	5,924	30,000	184,952

6 Related Work

EA analysis assesses certain criteria of an EA model [7]. By simulating an EA model, we can "enable enterprise architects to alter local strategies and observe their impact on the resulting global system behavior" [2]. Our proposal simulates an EA model in order to provide support for EA analysis by delivering information of the experiment execution to Business Analysts for their decision-making process. Taking into account the classification presented by Buckl et al in [3] for EA analysis, our proposal analyses *behavior statistics* and *dynamic behavior* for Body of Analysis dimension; *ex-post* and *ex-ante* for Time Reference; *indicator based* for Analysis Technique; and *non-functional* for Analysis Concern.

With this classification into consideration, we found various works with similar approaches. Among them, we found an approach based around system dynamics, agent-based and discrete events simulation [6]. They allow simulating elements from different domains or "views", which are of interest of distinct stakeholders. This is achieved by utilizing different simulation methods for each domain according to its simulation needs. Similarly, our approach supports the definition of metamodels for different domains, which can be composed into a single metamodel. But unlike their approach, our proposal uses only one simulation method in order to execute the EA model as a whole. Finally, similar to our approach, their proposal allows observing *behavior statistics*, *ex-post* and *ex-ante*, and *non-functional* analysis.

In [4], Crégut et al. animate models that conform to four fixed metamodels, where dynamism is expressed by defining execution states of the domain metamodel elements to control the progress of the overall animation. This approach is similar to out proposal, however, we represent dynamism using state machines that are external to the metamodel. This means that changes in the behavior of the elements, doesn't impact the metamodel and viceversa. Concerning simulation time management, [4] uses the states of elements to reflect time effects over the EA model, and time-passing information is recollected through events triggered by the elements of the model and expressed through event and traces metamodels, which are strongly connected to the EA metamodel and model. In contrast, our approach gathers information about the simulation using an observation model that is independent from the EA model. This represents an advantage since it leverages reusability of the model and metamodel. This approach focuses on both dynamic behavior and behavior statistics, as well as ex-post and ex-ante analysis, since the approach allows making decisions over the EA and adjusting the model to observe new behavior. Nevertheless, their approach doesn't specify how the animation results are presented in order to analyze them and make decisions.

Frank et al. [5] propose an indicator metamodel to evaluate non-functional properties of an organization within an Indicator System. Unlike our approach, they propose associating indicators to goals in order to assess if the system is supporting the organization's needs. Like our indicator language, they relate indicators to relevant elements of the organization, which are monitored to build indicators. On the other hand, Johnson et al. utilize Architecture Theory Diagrams (ATDs) [8] to assess the non-functional analysis dimension of the organization. We have studied this work and taken inspiration from it for our indicator language as well.

Finally, the main difference between our proposal and other EA simulation and analysis tools [4][5][6][8] is the flexibility for using ad hoc metamodels that include only domains of interest for the human experts, according to the project's needs. Other approaches define fixed metamodels that analysts are forced to use regardless of their own particular needs, which can introduce non-relevant information for the simulation into the models.

7 Conclusions

This paper proposes a model-based platform for building and simulating EA models to support decision making processes. The proposed platform offers the possibility of using EA metamodels, designed for particular projects, with any desired granularity, which is defined at the beginning of the project. This means that the metamodels only consider elements that pertain the EA analysis. The proposal also permits the definition of arbitrarily complex behaviors for each type of element involved, empowering the simulation. Additionally, allowing the definition of custom metamodels, we permit adapting the simulation to the particular definition of an organization, regardless of its business interests or requirements. This makes our proposal a suitable solution to make EA analysis for any enterprise.

Putting in place an EA simulation requires a number of different activities. Seeking to facilitate the definition of an EA simulation, we proposed an ideal workflow that aligns the simulation artifacts to the various activities of a simulation definition process. Furthermore, the characteristic of having multiple decoupled artifacts fosters reuse at several levels. One of those is the scenario/-experiment level: multiple experiments can be run on top of the same scenario, without requiring extensive modifications or re-designs. The results of all the experiments run over the same scenarios are presented using the same, comparable indicators. This facilitates the work of business analysts who participate in decision making processes and use the results of the simulation as inputs for analysis.

We have already considered several lines of future development for the platform. One of them is including complementing (and reusable) domain metamodels in the simulations. This should not be difficult to achieve on top of Cumbia since it has already shown its capacity to run multiple concern-specific workflow languages [10]. Additionally, although having custom-made metamodels can inccur in unnecessary extra effort, we plan to include means to transform existent EA metamodels and models designed with standard languages (e.g., BPMN or EPC), to facilitate the task of defining the simulation scenarios.

References

1. Banks, J., Carson, J.S., Nelson, B.L., Nicol, D.M.: Discrete Event System Simulation, 5th edn. Prentice Hall (2009)
2. Buckl, S., Matthes, F., Renz, W., Schweda, C.M., Sudeikat, J.: Towards simulation-supported enterprise architecture. In: Fachtagung Modellierung betrieblicher Informationssysteme (MobIS 2008), Saarbrucken, pp. 131–145 (2008)
3. Buckl, S., Matthes, F., Schweda, C.M.: Classifying Enterprise Architecture Analysis Approaches. In: Poler, R., van Sinderen, M., Sanchis, R. (eds.) IWEI 2009. LNBIP, vol. 38, pp. 66–79. Springer, Heidelberg (2009)
4. Crégut, X., Combemale, B., Pantel, M., Faudoux, R., Pavei, J.: Generative Technologies for Model Animation in the TopCased Platform. In: Kühne, T., Selic, B., Gervais, M.-P., Terrier, F. (eds.) ECMFA 2010. LNCS, vol. 6138, pp. 90–103. Springer, Heidelberg (2010)
5. Frank, U., Heise, D., Kattenstroth, H., Schauer, H.: Designing and utilising business indicator systems within enterprise models - outline of a method. In: Modelierung bietrieblicher Informationssysteme (MobIS 200) - Modellierung zwischen SOA und Compliance Management, Germany (2008)
6. Garret, C.: Understanding Enterprise Behavior using Hybrid Simulation of Enterprise Architecture. PhD thesis. Massachusetts Institute of Technology, USA (2009)
7. Johnson, P., Johansson, E., Sommestad, T., Ulberg, J.: A tool for enterprise architecture analysis. In: 11th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2007), p. 142 (October 2007)
8. Johnson, P., Nordstrom, L., Lagerstrom, R.: Formalizing analysis of enterprise architecture. In: Interoperability for Enterprise Software and Applications Conference, p. 10 (April 2006)
9. Sánchez, M., Jiménez, C., Villalobos, J.: Model based testing for workflow enabled applications. Computación y Sistemas (2011)
10. Sánchez, M., Jiménez, C., Villalobos, J., Deridder, D.: Extensibility in Model-Based Business Process Engines. In: Oriol, M., Meyer, B. (eds.) TOOLS EUROPE 2009. LNBIP, vol. 33, pp. 157–174. Springer, Heidelberg (2009)
11. Sánchez, M., Villalobos, J., Romero, D.: A State Machine Based Coordination Model applied to Workflow Applications. Avances en Sistemas e Informática 6(1), 35–44 (2009)
12. TICSW Research Group Universidad de los Andes. Banco de los Alpes. Website (August 2011), `http://sistemas.uniandes.edu.co/~losalpes/`
13. TICSW Research Group Universidad de los Andes. EA Simulation Engine. Website (January 2012), `http://cumbia.uniandes.edu.co/wikicumbia/doku.php?id=ea:start`