PROMASI — A PROject MAnagement SImulator

Nicholaos Petalidis
Voyager Software
Thessaloniki, Greece
petalidis@promasi.gr

Gregory Gregoriadis
Voyager Software
Thessaloniki, Greece
grigoriadis@promasi.gr

Alexandros Theodoridis
Informatics & Communications Dept (ICD)
TEI of Serres, Greece
theodoridis@promasi.gr

Antonios Chronakis
ICD
TEI of Serres, Greece
chronakis@promasi.gr

***Abstract*—This paper presents work-in-progress on a software project management simulator suitable for educating and training prospective software project managers. The work presents the design of a modular architecture for the simulator as well as a modeling language for representing possible management scenarios. It finally reports on issues one has to bear in mind when considering the development of such systems.**

***Keywords*-software engineering, software engineering education, project management simulation**

I. INTRODUCTION

Software engineering is generally considered a difficult subject to teach. Many authors have mentioned these difficulties (e.g. [1], [2]) and the software industry has complained several times that the level of education of future software engineers is not satisfactory [3], [4], [5].

An aspect of software engineering education that presents additional hurdles is that of project management because undergraduate students and other junior programmers frequently lack the level of experience required to grasp the importance of several project management concepts. Furthermore, the size of projects that can be exercised in a single semester does not justify many of the project management steps taught in the classroom while students more often than not will put emphasis on the technical aspects of the project and not on the required managerial procedures.

One approach used to overcome these problems is by use of simulation games in software project management courses (e.g. [6]). Simulation allows the student to see how realistic projects evolve and how his/her actions affect the outcome of the project.

To this end, this paper presents the design of such a system, i.e. a project management simulator (PROMASI-http://www.promasi.gr) that should allow students to actively test their skills in managing the development of a software product exercising various actions through simulation. One of the novel approaches of the solution proposed here is the separation of the various constituent parts of the simulator through a modular architecture allowing third parties to provide their own add-ons. We also present the development of extensions to the system dynamics modeling language [7] that will allow the construction of simulation models that can easily work with the rest of the PROMASI framework.

II. RELATED WORK

A number of simulators exist for the purposes of prediction or postmortem analysis of projects, but only few project management simulators have been developed for educational purposes.

Some of them are in essence system dynamic [7] models with limited interactivity, as in [6]. In these, the student sees a number of controls, in the form of buttons, sliders etc, that directly affect a number of project variables. Changing them, the system dynamics simulator generates new output and the student sees, in the form of graphs, the effect of his actions.

One problem with this approach is the reduced level of interactivity, since the students cannot easily change their choices after they receive feedback from the project. Another problem is that students do not have to go through the process of discovering the variables affecting the outcome, since these are presented directly to them. Another drawback is that the simulation model is tightly coupled to the user interface. This means that in order to simulate different scenarios or different parts of a project, it is necessary to write the simulator from scratch with a new user interface that takes account of any changes. Finally, simulators of this kind have too few of the characteristics that could classify them as "engaging" or captivating and that makes their adoption by the student community more difficult.

On the other hand, a small number of software project management simulators that improve the student experience with higher interactivity levels and decouple the user interface from the associated model are present.

SESAM [8] is one of the first simulators developed for educational purposes. It allows the definition of a static model that describes the attributes of the people working on a project, plus any related artifacts. The dynamic behavior of people and documents, including any influences from one to the other, is defined by the dynamic model. It is assumed that someone, not necessarily the teacher, will construct the static and dynamic models in such a way that they reflect to some extent the reality. In a typical situation then, the tutor will specify the attributes of the static model and the student will exercise the model by hiring or firing employees, talking to employees and the boss and assigning tasks among other things. SESAM uses its own language that allows the

definition of structures such as documents, persons and so on, the definition of relations between them and rules that define the conditions and the effect of allowable actions.

SimSE [9] is another simulator used for software project management courses. SimSE allows the definition of objects, actions and rules. An object is considered to be either an employee or an artifact or a tool or a project or finally a customer. The actions in a SimSE model represent the set of activities in which the objects in the simulation can participate and finally the rules describe the effects and the conditions for an action to occur. SimSE also uses its own language similar to that of SESAM. The language allows objects to be associated with graphics and rules maybe defined on when an action will trigger the appearance of a graphic. This way there is a separation between the user interface and the model and the same interface can be used with different models.

A rule-based engine with cause and effect rules is also used in SimVBSE [10] a slightly different simulator with emphasis on value-based software engineering. The rules are defined at design time and executed when the student plays. In a typical scenario the player is informed about various aspects of the organization he is working in, through a series of visits to different departments of the company and finally he is faced with a series of successive scenarios, an objective and a set of finite choices. After the student makes the choice he has the option to view a set of tutorials related to the scenarios he faced.

In all of the previous simulators the basic steps can be described as follows: construction of a model representing a particular process, instantiation of the model with specific values that represent a particular scenario, exercise of the model by a student who makes decisions at certain stages of the simulation.

A major difficulty in using these simulators however is in the construction of the models. Models need to be realistic, yet of educational value. Their construction is time-consuming and not something that can be done routinely in the course of a lecture. The above simulators all have their own languages but these languages are tightly integrated with the particular simulator and provide little guidance in the construction of realistic models. There are of course several other process modeling languages that can be used for describing a software development process, e.g. [11], [12], [13], [14] and ease the construction of a model. One of the problems of these languages, however, is that they are either not specific to software engineering or they are designed for postmortem analysis, or prediction or the specification of a particular scenario. Hence, they do not accommodate well the requirement for interactive scenarios with the user making decisions during the simulation. For this reason, we believe that they are not suited for use in an educational game simulation and we could not find any reported use of them in an educational setting.

III. The goals of ProMaSi

In order to avoid the problems that were presented above, the ProMaSi architecture is based on the following principles:

- Separation of the game and the model
  Decoupling of the user interface from the actual *business* logic is a standard software engineering practice. Similarly, the simulator must be playable with a variety of models and thus the interface that the user sees must be clearly separated from the underlying model.
- Independent modeling language
  The modeling language should be considered a separate part of the simulation game and should be defined as a separate entity that can be implemented in an autonomous component. This way the separation of concerns between the various simulator parts will be better achieved. Moreover the language should provide constructs specific for game-playing, i.e. the ability to define specific actions in the model where either input is expected or output will be generated. Later versions of the language should also provide a library of *components* that allow the quick creation of educational models. Such an extra layer is necessary in order to ease the construction of models. In this respect, we propose something similar to the language presented in [15], [16]. Like [15] we propose using system dynamics as the underlying language for defining a software process.
- Adequate set of models
  The construction of models is not generally an easy task and requires experience and skills that might not be readily available. One of the reasons, thus, that may be behind the low adoption rate of simulators in the class, is that there is simply not enough material to support a simulation-based software engineering management course. For this reason, it is critical that any simulator will come already equipped with a set of models that can be used in a course that follows generally accepted curriculum guidelines such as the Software Engineering 2004 volume [17]. Tools that will make the deployment of the simulator in lab environments easier and allow instructors to remotely provide assistance and guidance to students will also make the adoption of simulator-based training easier.

IV. ProMaSi Architecture

A. Internals

ProMaSi is organized into five layers, as presented in Figure 1 which implement different functionality and allow one to easily interchange them with different implementations. These layers are as follows:

Core The core engine of the simulator is a system dynamics model [7], [18]. ProMaSi extends the language of system dynamics so that interactive

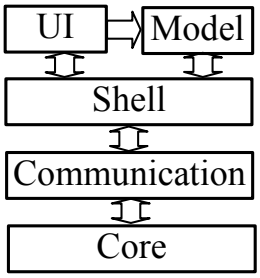

Figure 1. PROMASI's layers

simulation can take place by introducing events that can be communicated back and forth to the model. PROMASI's version of system dynamics is called PSD (PROMASI System Dynamics) and is briefly described below.

Communication
: Implements the API that provides input values to the core and sends the output values of the Core. The Core and the Communication layer can be used as the basis for any simulation game.

Model
: Implements the project management related entities such as *company*, *project* etc.

Shell
: Implements necessary APIs that allow different user interfaces to be connected to the simulator and different play modes (e.g. single user vs multi user) to be realized. It also provides the communication layer with any values requested by the model.

UI
: This layer implements the user interface. The current implementation simulates the desktop environment with appropriate tools, such as email, Gantt planner etc that one expects in a project manager's workstation.

When a game begins, the clock starts ticking and the actions of the project manager affect the Core execution, which in turn responds with appropriate events to the user. At each tick the Shell instructs the Core to perform a step. At every step the Core calculates the values of all the variables and outputs new values. When the Core requires an input value, it requests that value from the Communication layer and in turn the Communication layer requests that value from the Shell. The Core calculates the values and outputs the values to the Communication layer which forwards them to the Shell which updates the Model and the User Interface.

B. The modeling language

As mentioned earlier the basic system dynamics modeling language does not include any constructs for interactive simulation which is necessary for a playing experience. System dynamics include constructs for

Stocks
: Practically store the state of all interesting variables in the model (e.g. how much of the requirements is completed)

Flows
: Move information from one stock to another (e.g. the analysis flow may reduce requirement and increase analysis artifacts)

Variables
: Used whenever a constant is needed or a particular value needs to be temporarily stored.

Each stock is transformed by one or more flows (and vice versa), through explicitly defined functions (known as *calculated functions* in PSD). For example, the following:

$$requirements_t = requirements_{t=0} - \int_0^t analysis\ flow\ dt$$

defines how the requirement artifacts are transformed through the analysis flow. PROMASI system dynamics language supports most of the usual functions, constants, graphs, lookups etc found in such models.

In addition to the above constructs though, that are present in all system dynamic models, it also provides with extra language constructs that help realize the interaction with the other layers:

Output
: A construct with this characterization instructs the Core to send its value to the communication layer after every calculation

Events
: Every system dynamic object can have various events that are triggered when the specified conditions are met. When an event is triggered the Communication layer sends event information to the Shell. Each event looks like a typical function. If its value is calculated to be a number higher or equal to 1 then the event is triggered and it is sent to the upper layers. Each event can be triggered only once.

External
: A construct with this annotation instructs the Core to request the value for this variable from the communication layer.

Figure 2 shows a model drawn in the PROMASI editor. The structure shows how a bug fix proceeds: the amount of errors discovered (`errorDiscovery312`) depends on the team characteristics and the productivity of the individual working on the error (`averageTeamPlayerTest312`, `averageTesterTest312`). These are values that are calculated based on other variables whose value is provided by the Shell (*External variables*).

The graphical structure has an XML equivalent that looks as follows in the case of `bug312Progress`:

```
<void property="events">
 <void method="add">
```

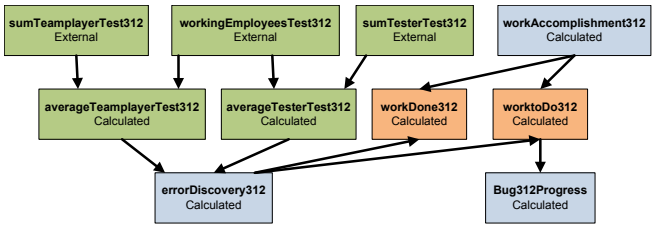

Figure 2. A PROMASI system model

```
<object class="Event">
 <void property="equation">
  <object class="CalculatedEquation">
   ...(truncated for brevity)
  <void property="equationString">
    <string>
     if(bug312Progress>=1,1,0)
    </string>
  </void>
...(truncated for brevity)
```

The XML shows for example that `bug312Progress` has an associated event that signals whenever progress goes over 100%.

C. Current status

The current version supports a single player score mode. In this mode the user plays alone and his actions are graded against an ideal score. The user plays a story which contains a series of projects that the user must complete. Stories are defined by the instructor in the form of XML documents. After each project concludes the user gets a score. The score is defined in the PSD model so the instructor can decide on the criteria that the score will be based on.

In a typical game scenario, after the user logs in, he selects a story from the set of stories that the instructor provided. After that he manages the project through typical project management tools such as Gantt charts and (obviously) email. Email notifications are generated by various events such as the initial project assignment or a particular task completion and they depend on what the underlying model has defined. Similar to typical project situations the user is given a specific budget and date to which the project must adhere. Figure 3 shows a screenshot of the typical situation the user finds himself into. In this screenshot, the email application is shown, where the user is informed about the project that was assigned to him (*Welcome to UBM...*). The marketplace is also shown where the user can choose from the available employees, having seen a brief bio of each one (*Robert Mash...*). Finally, the Gantt tool is shown where employee *Robert Mash* has been assigned the fix for *Bug312* and is given one day to complete it.

After the user concludes the project assignment the clock starts ticking until the next event takes place. The type of events that can take place depend on the complexity of the underlying model. In the end the user is shown a score as well as various important metrics of the project (for example the actual versus the originally assigned budget).

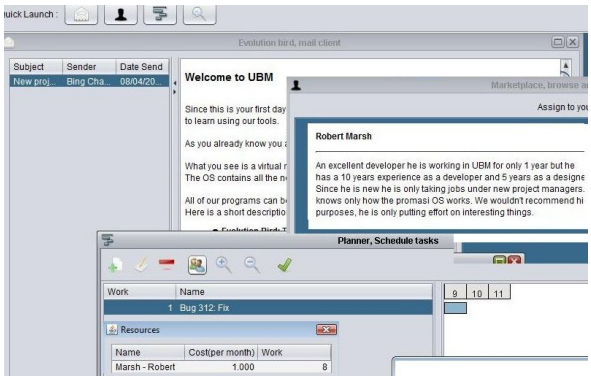

Figure 3. Screenshot of the PROMASI desktop

V. PROBLEMS UNCOVERED AND NEXT STEPS

The first evolution of the project has identified a number of areas that need special care when designing a simulator of this kind. Therefore, the next version of the project is scheduled to address these problems that are briefly described below:

A. Construction of realistic models

To be able to use the simulator as an efficient educational tool, proper models need to be created that give the user the impression of reality when dealing with various situations. At the moment, however, only simple proof-of-concept models have been developed. Work however is underway to provide models that take care of complex software project manager scenarios in situations where the user can exercise various alternatives. These alternatives include situations such as deciding on the number, duration and structure of iterations, deciding on whether they should hire or not more people etc.

B. Tools for model creation

Models should be constructed by the instructors but such models are not easy to create and may become quite complex. Thus tools are needed that help manage this complexity and allow the instructor to easily visualize the model, break it into sub-models and add or remove components. Furthermore, such model construction should be done in an integrated development environment that provides with other facilities as well such as source control and refactoring. To this end a new editor is being developed as an Eclipse plug-in so that the instructor will be able to build models in a familiar development environment.

C. Multi-user support

At the moment PROMASI only allows a single user to manage a single company. The user basically plays against an ideal market place which is simulated by the Core. This is not necessarily bad, but it has two drawbacks:

- Students need to be properly stimulated in order to actively engage in a course. Even though a game might be interesting on its own, competing against fellow students is certainly more interesting that competing against a machine
- Project assessment in the real world differs from theory. A project in theory might be a failure but in the real world might still be a success simply by the fact that it is better than its competitors.

For the above reasons a new line of development has been created that allows the simulator to be played concurrently by more than one user. In this setting every player manages his own company and hires the employees from a shared market place. Thus, players also compete against each other in order to hire the best employees. Every company undertakes the same projects and each company competes against the other in the market place.

In order to adopt the changes described above, the next version of PROMASI implements a standard client/server architecture where the Core layer executes on the server and communicates the simulation results whenever required to the clients.

VI. CONCLUSIONS

Simulators can be used to teach management concepts which would otherwise be difficult to show to undergraduate students. The design of such a simulator was presented here that provides with a modular architecture for easy customization and a modeling language suitable for interactive models. The first evolution of the project uncovered a series of issues that one needs to consider when designing such tools, the more important of which we believe is the creation of realistic models and the support for tools that will allow such a thing. PROMASI is addressing these problems in its next version. The interested reader can find more information on how to download the source or participate in the project, at the project's web site: http://www.promasi.gr

REFERENCES

[1] D. Evans, "Teaching software engineering using lightweight analysis," in *NSF CCLI Proposal*, 2001.

[2] D. Deveaux, R. Fleurquin, and P. Frison, "Software engineering teaching: a "Docware" approach," *SIGCSE Bulletin*, vol. 31, no. 3, pp. 163–166, 1999.

[3] M. Jazayeri, "The education of a software engineer," in *Proceedings of the 19th IEEE International Conference on automated software engineering*. IEEE Computer Society, 2004, pp. xviii–xxvii.

[4] D. Callahan and B. Pedigo, "Educating experienced IT proffessionals by addressing industry's needs," *IEEE Software*, vol. 19, no. 5, pp. 57–62, Sep./Oct. 2002.

[5] R. Conn, "Developing software engineers at the C-130J software factory," *IEEE Software*, vol. 19, no. 5, pp. 25–29, Sep/Oct 2002.

[6] D. Rodríguez, M. Ángel Sicilia, J. J. Cuadrado-Gallego, and D. Pfahl, "e-Learning in project management using simulation models: A case study based on the replication of an experiment," *IEEE Transactions on Education*, vol. 49, no. 4, pp. 451–463, 2006.

[7] J. Forrester, *Industrial Dynamics*. MIT Press, 1961.

[8] M. Deininger and K. Schneider, "Teaching software project management by simulation-experiences with a comprehensive model," in *Proceedings of the 7th SEI CSEE Conference on Software Engineering Education*. London, UK: Springer-Verlag, 1994, pp. 227–242.

[9] E. Navarro, "SimSE: A software engineering simulation environment for software process education," Ph.D. dissertation, School of Information and Computer Sciences, University of California, Irvine, 2006.

[10] A. Jain and B. Boehm, "SimVBSE: Developing a game for value-based software engineering," in *CSEET '06: Proceedings of the 19th Conference on Software Engineering Education & Training*, 2006, pp. 103–114.

[11] W. Emmerich and V. Gruhn, "FUNSOFT nets: A Petri-Net based software process modeling language," in *Proceedings of the Sixth International Workshop on Software Specification and Design*, 1991, pp. 175–184.

[12] S. Dami, J. Estublier, and M. Amiour, "APEL: A graphical yet executable formalism for process modelling," *Automated Software Engineering*, vol. 5, pp. 61–96, 1998.

[13] A. G. Cass, B. S. Lerner, S. M. S. Jr., E. K. McCall, A. Wise, and L. J. Osterweil, "Little-JIL/Juliette: a process definition language and interpreter," in *ICSE '00: Proceedings of the 22nd international conference on Software Engineering*, 2000, pp. 754–757.

[14] OASIS, *Web Services Business Process Execution Language Version 2.0*. OASIS, 2007.

[15] M. D. O. Barros, C. M. L. Werner, and G. H. Travassos, "A system dynamics metamodel for software process modeling," *Software Process: Improvement and Practice*, vol. 7, no. 3–4, pp. 161–172, 2002.

[16] M. de O. Barros, A. R. Dantas, G. O. Veronese, and C. M. L. Werner, "Model-driven game development: experience and model enhancements in software project management education," *Software Process: Improvement and Practice*, vol. 11, no. 4, pp. 411–421, 2006.

[17] I. C. Society, "Software engineering 2004," 2004, http://sites.computer.org/ccse/SE2004Volume.pdf.

[18] R. J. Madachy, *Software Process Dynamics*. IEEE Press, 2008.