A Software Process Simulator Machine for Software Engineering Simulation Games

Rafael O. Chaves[1] Emanuel M. da C. Tavares[2] Sandro R. B. Oliveira[2] Elói Luiz Favero[1]

[1]Postgraduate Program in Electrical Engineering – Instituto de Tecnologia – Universidade Federal do Pará (UFPA)

[2]Postgraduate Program in Computer Science – Instituto de Ciências Exatas e Naturais – Universidade Federal do Pará (UFPA)

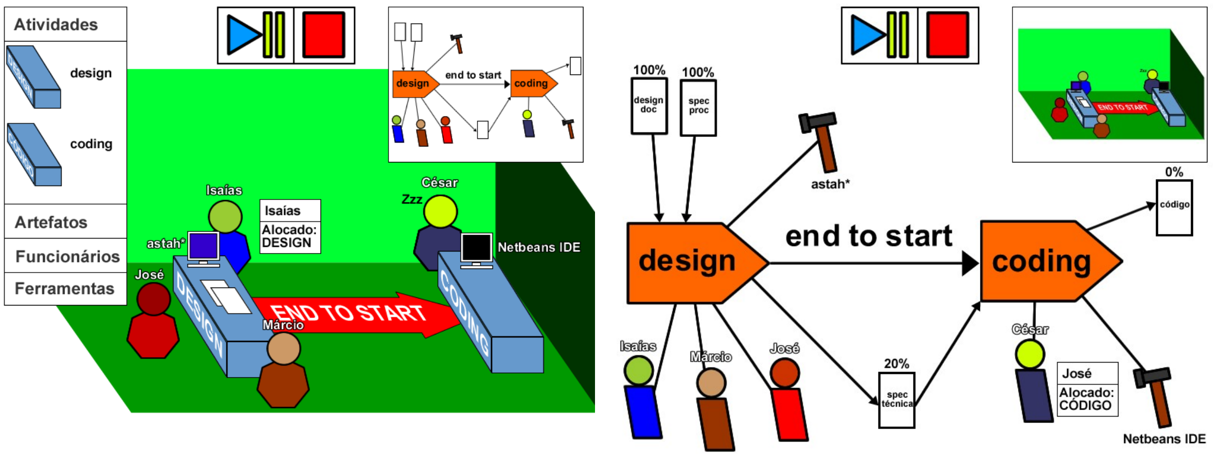


Figure 1: Conceptual screens from a "working in progress" game using the Software Process Simulation Machine

Abstract

Develop new games without having to start from scratch has been made possible by using game engines, since they offer a number of specialized components and optimized functions which are common in games. However, one realizes that process simulation games have not taken advantage of these technologies. This paper aims to demonstrate the results of the development of a software process simulation component called Software Process Simulator Machine (SPSM), which focuses on help and motivate software process educational game development through the addressing of main generic requirements for process simulation software.

Keywords: software, process, simulation, engines

Authors' contact:
`{emanuelmaues,rochaves,srbo,favero}@ufpa.br`

1. Introduction

Software components reuse is a common practice in much of the software industry. It offers advantages such as reducing the overall time and cost of development, higher quality and reliability in the product. Regarding the industry of game development, these components are known as game engines, and its reuse began in the late 80 [Anderson et. al. 2008].

It is possible to develop new games without having to start from scratch by using game engines because they offer a number of specialized components and optimized expected functions which are characteristics of a genre of game, e.g. Real Time Strategy (RTS) and First-Person Shooter (FPS) [Folmer 2007]. Thus, the development team can focus on specific aspects of the new game (e.g. plot, gameplay), since game common features, such as sound and treatment of collision are already implemented.

A game engine is usually designed for specific development of a genre of game. However, this exclusivity brings high costs to game design when using the engine to implement games from another genre [Anderson et. al. 2008].

Research in Software Engineering (ES) teaching has glimpsed the potential of using games to assist developers and project managers in both academical and professional ways. The application of simulation games to teach software process and software project management is primarily aimed to allow the trainee to virtually experience real situations that he could hardly practice during his academical life [Dantas et. al. 2004; Wangenheim and Shull 2009]. However, through literature review presented at section 5 of this paper, one realizes that, unlike genres as RTS and FPS games, process simulation games have not taken advantage of game engines or components specifically designed to ease and improve their development, particularities regarded.

This study is aimed to demonstrate the results of the development of a software process simulation component, called Software Process Simulator Machine (SPSM). Its implementation is based on simulation performance requirements and literature review on process simulation software. It is SPSM's intention to help and motivate software process educational game development through reusability, free software licensing and addressing the main generic requirements for process simulation software.

This paper is organized into five more sections: section 2 presents the motivation and relevance of this work, the third shows the architecture and requirements of SPSM, 4 describes and performs tests to validate the simulation, 5 describes related work and their relation with this article and 6 concludes with the results achieved and considerations about future work.

2. Motivation and Relevance

Education is one of the key goals of simulation [Oh 2006]. Regarding the conduct of research on software engineering simulation games, satisfactory results have been obtained, with some improvement of teaching, listed ahead: Hsueh [2008] found that these games, when employed for educational purposes, were well accepted by students; Dantas et. al. [2004] showed that they are suitable for testing situations that occur in project management; and, finally, Oh [2006] found that students successfully learn the concepts which the ES games addressed, however, concluded that they are more effective when they complement traditional forms of educating.

Various types of games have been proposed to improve ES teaching: card games [Baker et. al. 2003], board games [Taran 2007], quiz games [Wang et. al. 2007] and simulation games [Drappa and Ludewig 2000; Oh 2006; Dantas et. al. 2004]. The last two, in particular, are the most applied to ES education [Wangenheim and Shull 2009]. However, very little has been researched about components and architectural issues for its development; one of the few papers found which is concerned with these aspects is from Guedes [2006]. Generally, software process simulation machines, such as Hector [2001] and SESAM [2000], do not have natively support elements, structure and specific behaviors of software processes. In these machines, for each simulation model, the tutor shall program its elements, their relationships, attributes and behaviors.

Analogous to how games, from genres like RTS and FPS, receive support from specialized components to improve its development [Folmer 2007], it is expected that software process simulation games make similar use of these components.

3. Architecture and Requirements of Software Process Simulation Machine

The implementation of SPSM is an adaptation of eXtensible Software Process Engineering Meta-Model (xSPEM) [Brendaou et. al. 2007], aiming to expand the semantic of the software process models. Figure 2 shows the structure of the SPSM in the shape of a class diagram.

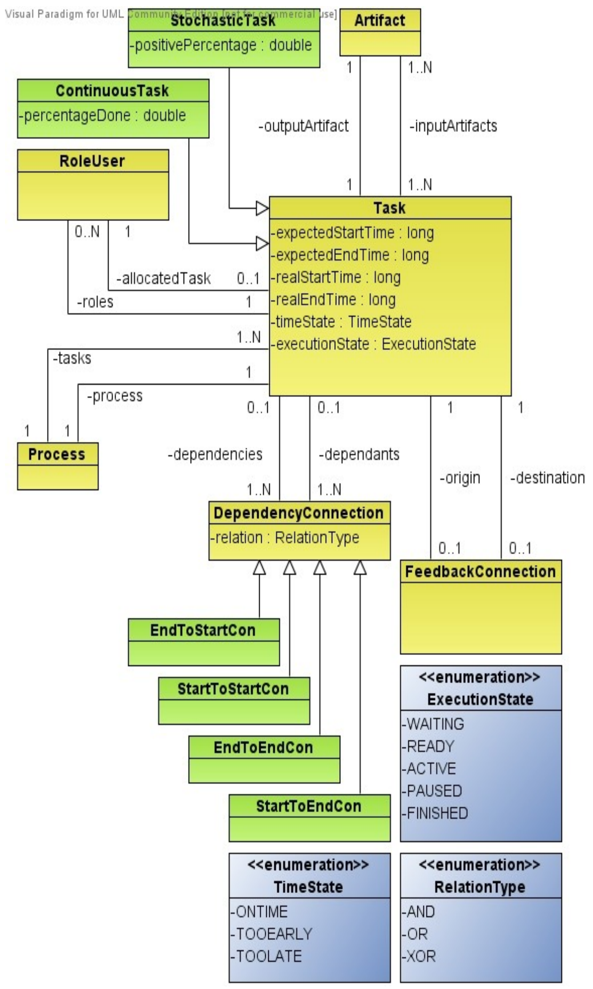


Figure 2. SPSM's architecture

Criteria is needed to define the requirements of the SPSM. These are:

1) Process Granularity and Approach: SPSM's simulation approach is discrete-event, as it focus on low-level details of the process [Acunha and Juristo 2005].

Processes is the granularity level chosen for SPSM, as it allows to notice the details of the software process [Zhang 2008], which is useful in a learning context. This granularity level has also been employed in other ES simulation games, such as SESAM [Draper and Ludewig 2000], SimSE [Oh 2006] and The Incredible Manager [Dantas et. al. 2004].

2) Software Process Elements: the software process' elements[1] were chosen so as to maintain the process' granularity level. In a sampling of process simulation games (e.g. SimSe [Oh 2006]) and software project management (e.g. TIM [Dantas et. al. 2004], SESAM [Ludewig and Draper 2000]), the following software process' elements were found in their simulation models (Table 1). These elements are divided between explicity defined, when it is already defined in the syntax of the process model, and implicity defined, where it must be defined, i.e. programmed by the user.

Table 1: Software process' elements (rows) and games that define them (columns). The letters I and E mean, respectively, Implicity Defined and Explicity Defined.

	SimSE	TIM	SESAM
Task	I	E	I
Role User	E	E	I
Artifact	E	E	I
Tool	E	E	I
Client	E	E	I
Project	E	E	I

3) Execution flow: SPSM defines what and when certain tasks are going to be executed according to precedence criteria and the choice between possible execution paths, establishing an execution flow. It is described below what are these criteria:

3.1) Dependency types: execution flow allowance, from start to the end of the process, is partially accomplished through the temporal order of precedence among tasks, established by its dependencies. Dependency types ground the simulation model to a closer real project implementation.

This requirement is based on xSPEM's dependencies [Brendaou et. al. 2007], specified in their enumerated class WorkSequenceKind, which is comprised of the following types: End to Start, End to End, Start to Start and Start to End. In order to expand the simulation semantics, these types have been converted into subclasses of Dependency. Beyond these types, FeedbackConnection was also defined (Figure 2), allowing that, given the outcome of a certain condition in a task, the execution flow returns to a previous task. FeedbackConnection is based on WebAPSEE's Feedback [Reis 2003].

3.2) Dependency relationships: summarized between two types:

Branch: an origin task depends on more than one destination task. Figure 3 show an example of this relationship.

1 Same as software process components [Softex 2009]

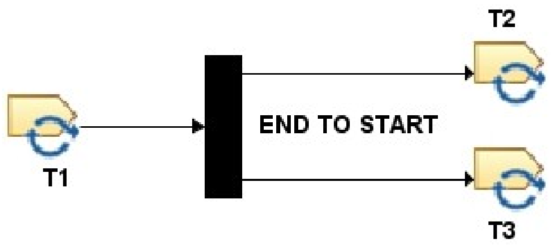


Figure 3. A Branch example. T1 is an origin task; T2 and T3 are destination tasks

Join: a destination task depends on more than one origin task. Figure 4 gives an example.

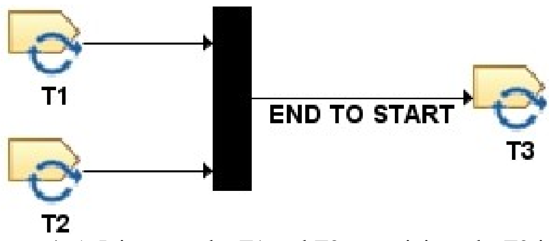


Figure 4. A Join example. T1 and T2 are origin tasks;T3 is a destination task.

Dependency relationships were based on BPMN's dependencies (gateway): AND, OR and XOR [2010]. They behave like a Branch (Decision, Parallel Forking) or like a Join (Merge, Parallel Joining). Both Branch and Join possess a logical operator, e.g. true or false, that allows the execution path to choose the next tasks to be executed.

4) Simulation parameters: attributes of the process' elements that establish the temporal relationship between tasks and define their execution states. This SPSM requirement is adapted from xSPEM's package xSPEM_ProcessObservability [Brendaou et. al. 2007]. The adjustments are listed below:

1 – ExecutionState class (Figure 2) shares the same types of ActivityState from xSPEM, with the addition of Ready type, which defines that a certain task can become Active when all its dependencies are satisfied or simply do not exist.

2 – Activity class from xSPEM, defined as Task in SPSM, is branched in two subclasses:

2.1 – Continuous Task: It is a task in which its execution duration (range between expectedStartTime and expectedEndTime) is measured with a percentual value.

2.2 – Stochastic Task: A origin task in which its execution flow might proceed to one of two destination tasks depending on the outcome of a probabilistic decision that varies between true or false.

The definition of a task's time state in relation to the simulation's overall time comes from TimeState class, that shares the same states of ActivityTime class from xSPEM: tooEarly, tooLate and onTime. The state informs if a task is delayed, rushed or running exactly on time after checking the difference between the expected start/end times and real start/end times.

5) Adherence to the SPIDER_ML syntax: Moody [1994] suggests that graphical modeling tools can be used to partially model process that will be executed by a process engine. Huff [1996] states that process modeling languages have been developed with syntax constructors specifically designed to address software process' particularities.

The employment of process modeling languages manages to abstract only the relevant points of the process model that will be executed, which eases its definition. Regarding this paper, it was decided that the simulated process must be, first, described using the software process modeling tool SPIDER_PM, which uses SPIDER_ML as its process modeling language [Oliveira 2009]. Reasons for SPIDER_ML's usage in this work concern it being a profile of SPEM 2.0 [OMG 2010] and have almost all process' elements necessary for simulation defined at processes granularity level. Table 2 shows the relationship between SPIDER_PM's defined elements and SPSM's.

Table 2. Comparison between SPIDER_PM's defined elements and SPSM's.

	SPIDER_PM	SPSM
Task	Defined	Defined
Role	Defined	Defined
Artifact	Defined	Defined
Tool	Defined	Not defined
Client	Not defined	Not defined
Project	Defined	Defined

It is emphasized that SPSM's main objective is to simulate software processes according to the criteria defined above. The behavior and interactions of resources allocated to the process (e.g. system dynamics, intelligent agents [Guedes 2006]) are outside its scope.

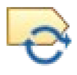
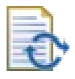
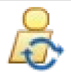
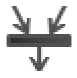
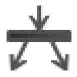
While SPSM contemplates most of the component elements and requirements for implementing software process to provide a greater realism to the simulation, its accuracy is not sufficient to simulate evaluative or predictive models. At its current stage, its only purpose is education and training, according to Kellner et. al. [1999] classification.

4. Tests

To demonstrate the SPSM capabilities as a process simulator component, tests were conducted to validate the following simulation aspects: execution flow, syntatic correctness and process instantiation. These tests consist of software processes that are simulated by SPSM to validate its established requirements and features. As a consequence of size limitation, only tests with at most three tasks are presented.

The simulated software processes' graphical notation is SPIDER_ML. Table 3 describes the modeling language's elements applied in this work.

Table 3. List of SPIDER_ML elements employed in this work. Adapted from Oliveira [2009].

Name	Graphical Notation	Description
Instantiated Task		A defined task in the process which has information about when they will be initiated, completed, its type and its contribution to the process.
Instantiated Artifact		Work products that are consumed, modified and produced by instantiated tasks.
Instantiated Role		A process' actor with capabilities and skills that enable him to perform or assist in the execution of tasks.
Start Point	●	The starting point of the simulated process.
End Point	◉	The finishing point of the simulated process.
Join		Allows the execution flow to converge from one to several tasks. It has a logical operator which determines the execution path.
Branch		Allows the execution flow to converge from several to one task. It has a logical operator which determines the execution path.
Dependency	↘	Represent the dependencies among the tasks.
Condition	◇	Represents the stochastic task's condition.

In order to meet the SPSM requirements, tests were separated among three categories: 1) execution flow, destined to verify if the flow converges to the expected tasks, regarding its dependencies only; 2) syntactic correctness, to detect when the simulated process' syntax does not meet the requirements of the simulation; and

3) instantiation, to verify the behavior of the simulated tasks when it has and has not allocated resources. It is noteworthy that, at the first category of tests, role users and artifacts are shown for the sole purpose of syntactic correctness, although they are not relevant at that context.

4.1 Execution Flow

Three processes are modeled for this category: one with a branch structure, another with a join and the last with a feedback loop, as described below:

1) The first process involves three tasks: T1, which is the origin; T2 and T3, both as destination tasks. There is an End to Start dependency between origin and destination, establishing a Branch structure. The logical operator is AND, meaning that, when T1 finishes, both T2 and T3 shall start. Figure 5 shows the simulated process in SPIDER_ML syntax.

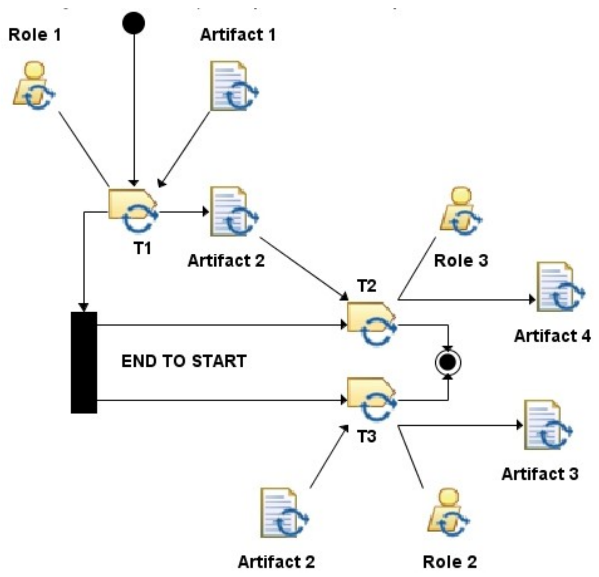


Figure 5. First simulated process modeled in SPIDER_ML.

The duration of each task is specified by subtracting the actual finishing time and the actual starting time, as demonstrated in (1). Table 4 presents the values assigned to the tasks.

duration = actual finishing time – actual starting time (1)

Table 4. Values assigned to the tasks from the first simulated process.

	Actual Starting Time	Actual Finishing Time	Duration
T1	0	3	3
T2	3	6	3
T3	3	7	4

Most of the line graphs presented in this section demonstrate the simulated process' execution flow employing homogeneous colored bars. Each color represents an execution state. Table 5 summarizes them:

Table 5. Execution states (columns) and their respective colors (rows)

Color	Execution State
Blue	Waiting
Orange	Ready
Yellow	Active
Red	Finished

Figure 6 shows the simulation's results in the shape of a line graph. X Axis presents the overall process time while Y Axis shows each task and their respective execution states according with time.

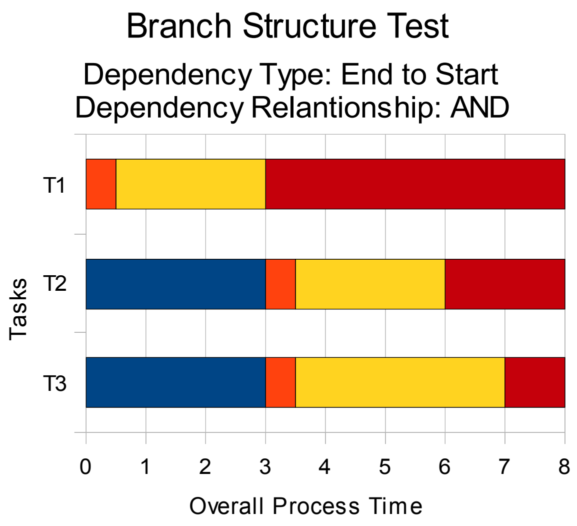


Figure 6. T1, T2 and T3 task progress chart of the first test.

The task checks its status and evaluates the logical conditions to change its state more than once per unit of overall. Acknowledging this, the task state will be modified whenever the logical conditions are met, without waiting for the addition of a unit of overall time. Figure 6 demonstrates this behavior, since tasks T1, T2 and T3 exchange state Ready to Active in the same unit of time.

2) The second test involves three tasks: T1, T2 and T3. T1 and T2 are origin tasks and T3 are destination tasks. Between them, there is an End to Start dependency type, establishing a Join Structure. The logical operator is AND, meaning that, when T1 and T2 finishes, T3 shall start. Figure 7 shows the simulated process in SPIDER_ML syntax. The duration assigned to T1, T2 and T3 is shown in Table 6. Figure 8 shows the simulation's second test result using the same line graph from the first test.

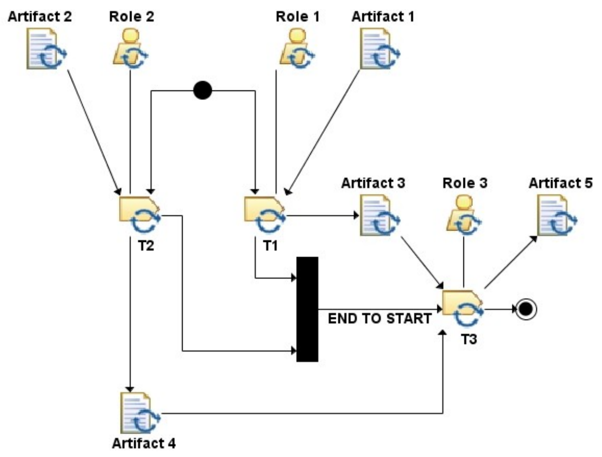


Figure 7. Second simulated process modeled in SPIDER_ML.

Table 6. Values assigned to the tasks from the second simulated process.

	Actual Starting Time	Actual Finishing Time	Duration
T1	0	3	3
T2	0	3	3
T3	3	7	4

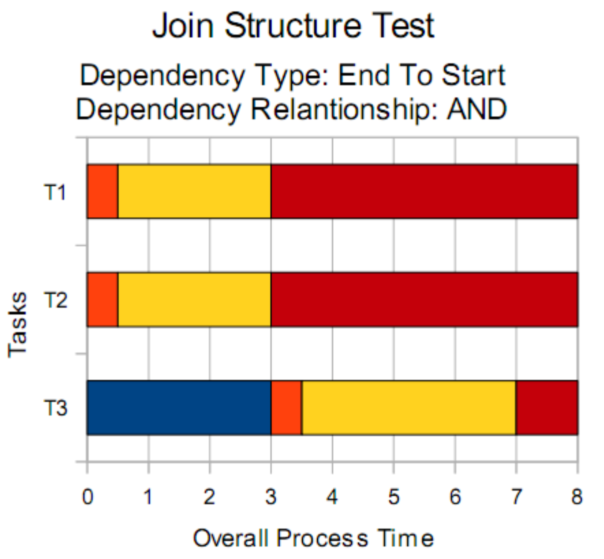


Figura 8. T1, T2 and T3 task progress chart of the second test.

3) The third process has T1 as a origin, T2 as a destination and a End to Start dependency between them. However, T2 is a stochastic task, meaning that, if the condition result is “yes”, the process ends. A “no” forces a return to T1 throught a feedback connection, changing its state to “Active”. Figure 9 presents the process modeled with SPIDER_ML. Table 7 shows the assigned values for each task present at this test.

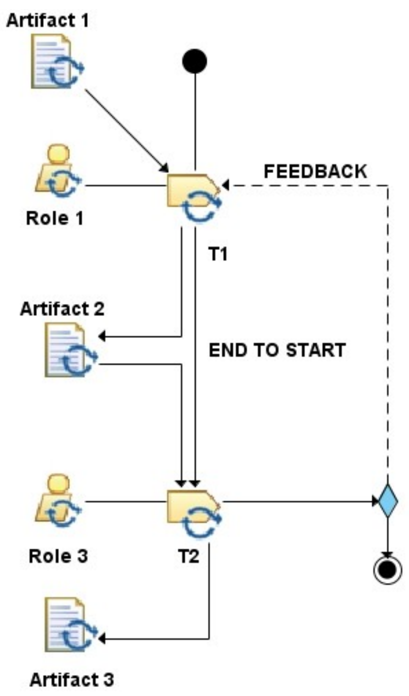


Figure 9. Third simulated process modeled in SPIDER_ML.

Table 7. Values assigned to the tasks from the third simulated process.

	Actual Starting Time	Actual Finishing Time	Duration
T1	0	1	1
T2	0	1	1

The expected outcome of this test is that, as long as T2 condition results in “no”, the flow will keep returning to T1, until it gives an “yes”. Figure 10 shows the task behaviors during the test. It is noteworthy that the first T2 condition result was “no”, when the overall process time was “1”. The second time, it resulted in an “yes”, when the overall process time was “3”.

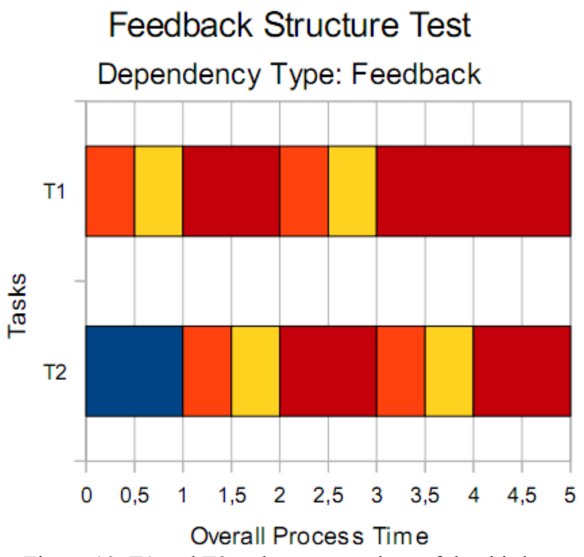


Figure 10. T1 and T2 task progress chart of the third test.

4.2 Syntactic Correctness

At this category, tests focus to demonstrate SPSM ability to detect syntax errors inside the simulated models. First, it is modeled a task devoid of input and output artifacts. Second, its behavior is studied.

The test is modeled to comprise three possibilities: task T1 has no input and output artifacts (Figure 11.a); only has inputs (Figure 11.b); has both (Figure 11.c). Table 8 presents the assigned values of the task. The expected result for this test is that only when T1 has both input and output artifacts connected, its state changes to Active.

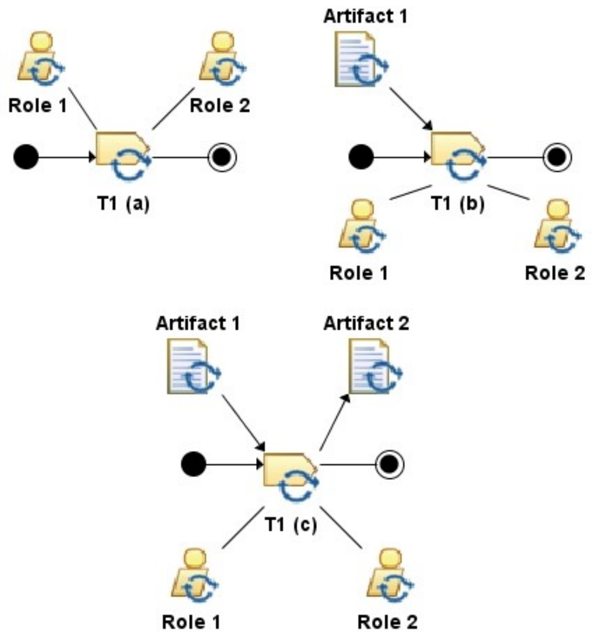


Figure 11. Fourth simulated processes modeled in SPIDER_ML.

Table 8. Values assigned to the tasks from the fourth simulated process.

	Actual Starting Time	Actual Finishing Time	Duration
T1	0	3	3

Figure 12 shows the test performed at a single process time unit. Four situations are summarized: the green bar shows a task with no input or output artifact, blue represents a task with no output, orange is a task with no input and, finally, yellow demonstrates a task with at least one input and one output artifact. The Y axis shows the task progress for each time unit of the overall process time. To address a better understanding, Figure 12 shows the green, blue and orange bar below the ordinate axis to demonstrate that the absence of an input or an output artifact will make the task to stop progressing.

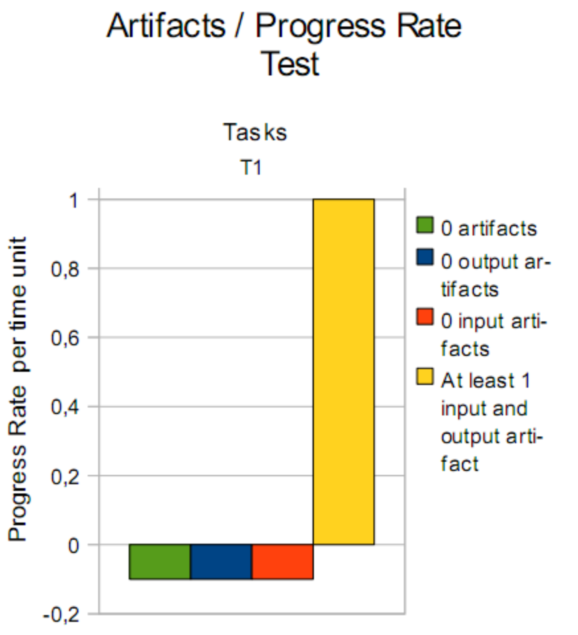


Figure 12. Performance of T1 based on the number of artifacts owned in a single unit of time.

Figure 13 represents the performance of the task / number of artifacts in relation to the overall process time, represented in the X axis. The red line shows the progress of the task. The yellow and blue bars represent, respectively, the number of input and output artifacts owned by the task. The Y axis serves to represent both task execution time and its number of artifacts.

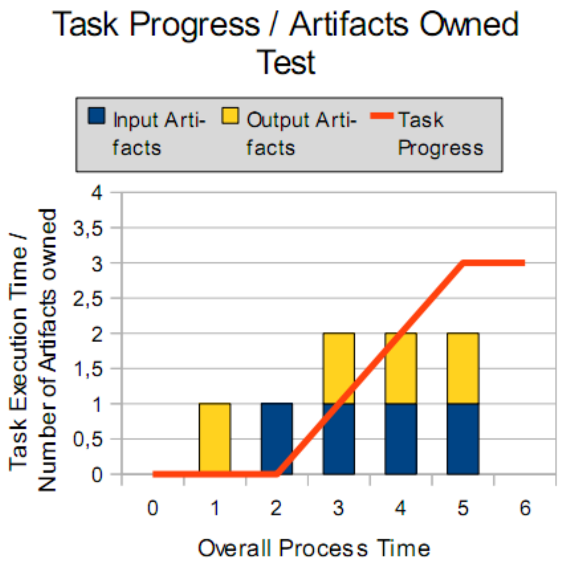


Figure 13. Task progress and its relation with input and output artifacts.

4.3 Process Instantiation

This test category wants to validate the requirement that a task can only be initiated, i.e. its state can change to active, when it has at least one role user assigned. If this role is deallocated before the task finishes, its state will change to paused. The importance of this requirement is to make the resource allocation more flexible.

A process containing only one task is modeled. It considers three possible situations: a task with no roles (Figure 14.a), with only one (Figure 14.b), with more than one (Figure 14.c). The expected result is that only when T1 has at least one assigned role user, its task state changes to "Active". Table 9 shows the assigned values of T1's attributes.

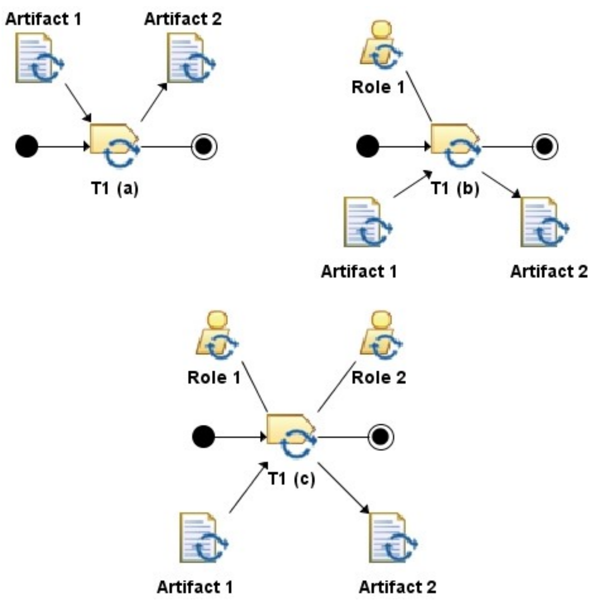


Figure 14. Fifth simulated processes modeled in SPIDER_ML.

Table 9. Values assigned to the task from the fifth simulated process.

	Actual Starting Time	Actual Finishing Time	Duration
T1	0	3	3

The line graph below (Figure 15) shows the test performed at a single time unit process. Its bars represent one possibility each: the green bar is the progress rate of a task devoid of role users; the blue bar is the exact opposite, as long as it has an allocated role user. The Y axis shows the progress of the task for each process time unit. To address a better understanding, Figure 15 shows the green bar below the ordinate axis to demonstrate that the absence of an assigned role user will make the task to stop progressing.

The next line graph (Figure 16) represents the task progress along the overall process time (X axis) accordingly with its assigned role users. The red line shows the task progress. The blue bar is the number of the task's allocated roles. The Y axis serves both to represent the task progress and the number of assigned roles.

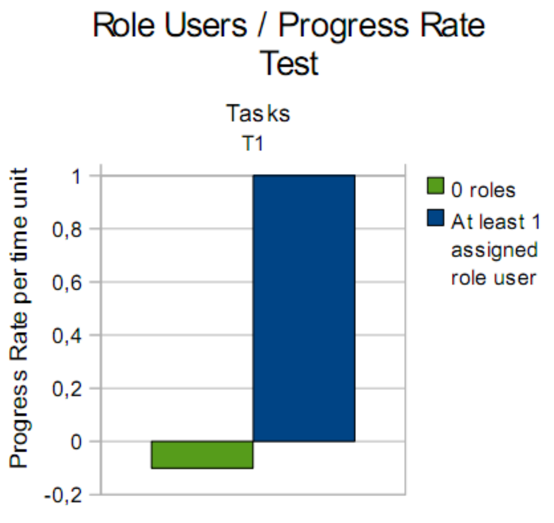


Figure 15. Performance of T1 based on the number of assigned roles in a single unit of time.

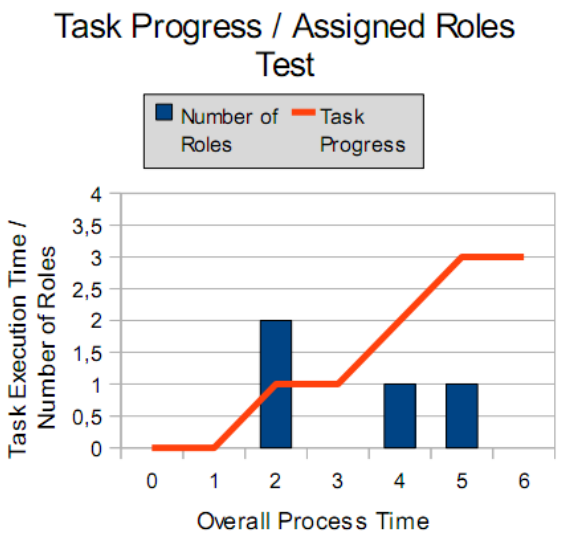


Figure 16. Task progress and its relation with the assigned roles.

5. Related Works

Currently, there are two software process simulator machines for educational purposes cited by Wangenheim and Schull [2009], named SESAM Simulator and Hector. Table 10 presents a comparison between these and SPSM requirements.

1) SESAM Simulator: this simulator takes as input a software process model comprised of: 1) a static model, which creates and specifies process' elements and their relationships; and 2) the rules which specifies the process behavior through native conditions and their consequences to the simulation.

2) Hector: is a software process simulation environment that translates a Process Modeling Language [Barros 2001] into System Dynamic constructors. Two types of complementary models are used: 1) domain model, which describes process' elements, their relationships, attributes and behaviors through an objectoriented paradigm approach; and 2) instantiated model, which instantiates the domain model's elements, resulting in the creation of a specific software process. Several instantiated models can be created from the same domain model. Hector is responsible for the simulation aspects of The Incredible Manager [Dantas et. al. 2004] and Guedes' MsC thesis [2006].

Table 10. Comparison between SESAM, Hector and SPSM requirements.

	SESAM	HECTOR	SPSM
Approach and Granularity Level	Process	Process	Process
Elements	Possible to define	Possible to define	Already defined
Dependency Types	Finish to Start	Finish to Start	Finish to Start Start to Finish Start to Start Finish to Finish Feedback
Dependency Relationships	AND	AND	AND OR XOR
Division between model and simulator	Process must have the same domain of the simulator	Process must have the same domain of the simulator	Any process model
Adherence to a Process Modeling Language	No	No	Yes
Simulation Paradigm	Discrete Event	Discrete Event	Continuous Discrete Event

6. Conclusions

SPSM is generic enough to simulate most of the execution flow that may occur in real software processes. However, it is still unable to check for inconsistencies in the process model, since there are many process modeling possibilities. In spite of this, SPSM provides a high level abstraction for the process definition, reducing the time spent describing it, but it brings the disadvantage of not allowing the creation of new elements, attributes and states beyond those already defined in the existing model.

Currently, two software process simulation games are being developed employing SPSM. Those are: SiMPS, a Software Process Improvement Simulation game, which focus on process improvement training; and EnactMe (Figure 1), that aims to give a complementary training to the process' role users through simulation.

Authors presented the most relevant tests to demonstrate SPSM application as a software process simulation component, although no tests were performed to verify the performance scale of the simulation. A graphical interface for SPSM is currently being developed as future work.

Acknowledgements

The authors would like to thank the Project Spider [Oliveira 2009] staff by their collaboration at modeling the processes applied as tests in this paper.

References

Acunha, S. T. and Juristo N. (Eds.), 2005. *Software Process Modeling*. Springer Press.

Anderson, E. F., et. al., 2008. The case for research in game engine architecture. *Proceedings of the 2008 Conference on Future Play: Research, Play, Share.*

Baker, A., et al., 2003. An Experimental Card Game for Teaching Software Eng. *Proceedings of the 16th Conf. Software Eng. Education and Training*, IEEE CS Press, pp. 216–223.

Barros, M de O., 2001. Gerenciamento de Projetos baseado em Cenários: Uma Abordagem de Modelagem Dinâmica e Simulação. Thesis (PhD) held in the Postgraduate Program in Engineering Systems and Computing at UFRJ.

Brendaou, R., et. al., 2007. Definition of an Executable SPEM 2.0. *Procedings of the 14th Asia-Pacific Software Engineering Conference,* IEEE CS Press, pp. 390–397.

Business Process Management Iniciative. Available from: http://www.bpmn.org/ [Acessed 13 July 2010].

Dantas, A. R., et. al.., 2004. A Simulation-Based Game for Project Management Experiential Learning. *Proceedings of SEKE 2004*, pp. 19-24.

Drappa, A. and Ludewig, J., 2000. Simulation in Software Engineering Training. *Proc. 22th Int'l Conf. Software Eng.*, ACM Press, pp. 199–208.

Folmer, E., 2007. Component based game development: a solution to escalating costs and expanding deadlines?. *Proceedings of the 10th international conference on Component-based software engineering*.

Guedes, M. S., 2006. *Um Modelo Integrado para Construção de Jogos de Computador aplicado à Capacitação em Gestão de Projetos*. Thesis (MsC) held in the Postgraduate Program in Computer Science at UFPE.

Hsueh, N., et. al., 2008. Applying UML and software simulation for process definition, verification, and validation. *Information and Software Technology*.

Huff, K. In: Fuggetta, A. and Wolf, A. (Eds.), 1996. Software Process. John Wiley & Sons.

Kellner, M. I., et. al.., 1999. Software process simulation modeling: Why? What? How?. *Journal of Systems and Software*, v. 46, pp. 91-105.

Moody, S. A., 1994. The STARS process engine: language and architecture to support process capture and multi-user execution. *Proceedings of the conference on TRI-Ada '94*.

Object Management Group. Available from: http://www.omg.org/ [Acessed 12 July 2010].

Oh, E. N., 2006. *SimSE: A Software Engineering Simulation Environment for Software Process Education.* Thesis (PhD) held in the Postgraduate Program in Information and Computer Science at University of California, Irvine.

Oliveira, S. R. B., 2009. SPIDER - Uma Proposta de Solução Sistêmica de um Suite de Ferramentas de Software Livre de apoio à implementação do modelo MPS.BR. Research Project. Instituto de Ciências Exatas e Naturais, Universidade Federal do Pará, Belém.

Reis, C. A. L., 2003. Uma Abordagem Flexível para Execução de Processos de Software Evolutivos. Thesis (PhD) held in the Postgraduate Program in Computer Science at UFRGS.

Softex, 2009. Guia Geral do MPS.BR. Available from: http://www.softex.br/mpsbr/_guias/guias/MPS.BR_Guia_Geral_2009.pdf. [Acessed 12 July 2010].

Taran, G., 2007. Using Games in Software Engineering Education to Teach Risk Management. *Proc. 20th Conf. Software Eng. Education and Training*, IEEE CS Press, pp. 211–220.

Wang, A. I., et. al., 2007. LECTURE QUIZ - A Mobile Game Concept for Lectures. *The 11th IASTED International Conference on Software Engineering and Application. SEA 2007*, Cambridge, Massachusetts, USA..

Wangenheim, C. G. V. and Shull, F. 2009. To Game or Not to Game. *IEEE Software*, vol. 26, no. 2, pp. 92-94.

Zhang, H., et. al., 2008. Software Process Simulation Over the Past Decade: Trends Discovery from a Systematic Review. *Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement*.